# Using NROFF and TROFF

Part Number: 800-1755-10
Revision A, of 9 May 1988

Material in this manual comes from a number of sources: *Nroff/Troff User's Manual*, Joseph F. Ossanna, Bell Laboratories, Murray Hill, New Jersey; *A Troff Tutorial*, Brian W. Kernighan, Bell Laboratories, Murray Hill, New Jersey; *Typing Documents on the UNIX System: Using the -ms Macros with Troff and Nroff*, M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *A Guide to Preparing Documents with -ms*, M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *Document Formatting on UNIX Using the -ms Macros*, Joel Kies, University of California, Berkeley, California; *Writing Papers with Nroff Using -me*, Eric P. Allman, University of California, Berkeley; and *Introducing the UNIX System*, Henry McGilton, Rachel Morgan, McGraw-Hill Book Company, 1983. These materials are gratefully acknowledged.

# Contents

# Tables

# Figures

# Preface

This manual provides reference information and examples for the text formatters `nroff` and `troff`. We assume you are familiar with a terminal keyboard and the Sun system. If you are not, see *Getting Started with SunOS: Beginner's Guide* for information on the basics, like logging in and the Sun file system. If you are not familiar with text editors, read *Doing More with SunOS: Beginner's Guide* and the chapter "Introduction to Text Editing" in *Editing Text Files*. Finally, we assume that you are using a Sun Workstation, although specific terminal information is also provided.

For additional details on Sun system commands and programs, see the *SunOS Reference Manual.*

## Summary of Contents

Here is a summary of the chapters that follow:

1. *Introduction* — Describes what `troff` can do for you, some tools you can use with `troff` or `nroff` to refine your results, how to use `nroff` and `troff`, the differences between the two text formatting programs, and a little about the mechanisms built-in to `nroff` and `troff`.

2. *Line Format* — Explains how the text formatting programs fill and adjust text input lines and how various formatting requests affect filling and adjusting functions in `troff`.

3. *Page Layout* — Describes the default page layout parameters built-in to `troff` and how you can alter them. Also explains how certain formatting requests interact in laying out pages.

4. *Line Spacing and Character Sizes* — Explains the available type and spacing sizes in `troff` and `nroff`, and how to change them.

5. *Fonts and Special Characters* — Describes the fonts available with `nroff` and `troff` and how to change them.

6. *Tabs, Leaders, and Fields* — Explains what tabs, leaders, and fields are, and how to set them.

7. *Titles and Page Numbering* — Explains how to create page headers and page footers. Also covers how to use the built-in `troff` page number register to print page numbers on your document automatically.

8. troff Input and Output — Describes how to embed files within files, to switch input from one file to another, to display a message on your terminal when troff reaches a certain point in a file, and in nroff only, how to pipe the output from a file to a program by using a special nroff command in the file.

9. *Strings* — Explains how to give a string of characters a new name so you can reference them easily. Also provides a facility for referencing the values of the strings.

10. *Macros, Diversions, and Traps* — Describes how to define macros, store information in diversions, and use diversions and traps to process text at specific places on pages.

11. *Number Registers* — Explains what troff number registers are and what you can use their values for.

12. *Drawing Lines and Characters* — Describes the several built-in troff functions for moving to arbitrary places on the page and for drawing things.

13. *Character Translations* — Describes how to change the escape character and translate the value of one character into another.

14. *Automatic Line Numbering* — Explains how to use the troff requests for numbering lines in the output file.

15. *Conditional Requests* — Describes troff mechanisms for conditionally accepting input.

16. *Debugging Requests* — Explains requests for displaying names and sizes of defined macros, flushing the output buffer, and aborting the formatting.

17. *Environments* — Describes how to shift input processing between the three nroff/troff environments.

A. troff *Request Summary* — A quick reference summarizing nroff and troff requests.

B. *Font and Character Examples* — Several tables of special characters like Greek letters, foreign punctuation, and math symbols.

C. *Escape Sequences* — Summarizes escape sequences for obtaining values of number registers, for describing arbitrary motions and drawing things, and for specifying certain miscellaneous functions.

D. *Predefined Number Registers* — Tables of troff General and Predefined Number Registers

E. troff *Output Codes* — A summary of the binary codes for the C/A/T phototypesetter.

### Conventions Used in This Manual

Throughout this manual we use

```
hostname%
```

as the prompt to which you type system commands. **`Boldface typewriter font`** indicates commands that you type in exactly as printed on the page of this manual. `Regular typewriter font` represents what the system prints out to your screen. Typewriter font also specifies Sun system command names (program names) and illustrates source code listings. *Italics* indicates general arguments or parameters that you should replace with a specific word or string. We also occasionally use italics to emphasize important terms.

### Notation Used in This Manual

Numerical parameters are indicated in this manual in two ways. ±*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 lower case letters and one- or two-character arguments are indicated by a pair of lower case letters. Character string arguments are indicated by multi-character mnemonics.

# 1

# Introduction

# 1

# Introduction

## 1.1. nroff and troff

nroff and troff are text processing utilities for the Sun system. nroff formats text for typewriter-like terminals (such as Diablo printers). troff is specifically oriented to formatting text for a phototypesetter. nroff and troff accept lines of text (to be printed on the final output device) interspersed with lines of format control information (to specify how the text is to be laid out on the page) and format the text into a printable, paginated document having a user-designed style. nroff and troff offer unusual freedom in document styling, including:

- detailed control over page layout;
- arbitrary style headers and footers;
- arbitrary style footnotes;
- automatic sequence numbering for paragraphs, sections, etc;
- multiple-column output;
- dynamic font and point-size control;
- arbitrary horizontal and vertical local motions at any point;
- a family of automatic overstriking, bracket construction, and line drawing functions.

nroff and troff are highly compatible with each other and it is almost always possible to prepare input acceptable to both. The formatters provide requests (conditional input) so that you can embed input expressly destined for either nroff or troff. nroff can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal.

This manual provides a user's guide and reference section for nroff and troff. Note that throughout the text we refer to nroff and troff more or less interchangeably — places where the narrative refers specifically to one or the other processor are noted.[1]

You should be aware that using nroff or troff 'in the raw' requires a detailed knowledge of the way that these programs work and a certain knowledge

[1] The material in this chapter evolved from *A troff Tutorial*, by Brian Kernighan of Bell Laboratories, and from *nroff/troff User's Manual*, originally written by Joseph Ossanna of Bell Laboratories.

of typographical terms. nroff and troff don't do a great deal of work for you — for example, you have to explicitly tell them how to indent paragraphs and number pages and things like that.

If what you are trying to do is just get a job done (like writing a memo), you shouldn't be reading this manual at all, but rather the chapter "Formatting Documents with the -ms Macros" in the *Formatting Documents* manual. If, on the other hand, you would like to learn the fine details of a programming language designed to control a typesetter, this is the place to start reading.

In many ways, nroff's and troff's control language resembles an assembly language for a computer — a remarkably powerful and flexible one — many operations must be specified at a level of detail and in a form that is too hard for most people to use effectively.

The single most important rule when using troff is not to use it directly, but through some intermediary such as one of the macro packages, or one of the various preprocessors described in *Formatting Documents*. In the few cases where existing macro packages don't do the whole job, the solution is *not* to write an entirely new set of troff instructions from scratch, but to make small changes to adapt existing packages. In accordance with this strategy of letting someone else do the work, the part of troff described here is only a small part of the whole, although it tries to concentrate on the more useful parts. In any case, there is no attempt to be complete. Rather, the emphasis is on showing how to do simple things, and how to make incremental changes to what already exists. If you are interested in the complete story, look into the troff source itself.

## Text Formatting Versus Word Processing

Many newcomers to the UNIX system are surprised to find that there are no word processors available. This is largely historical — the types of documents (such as the Sun manuals) that people do with the UNIX system's text formatting packages just can't be done with existing word processors. Before you get into the details of nroff and troff, here is a short discussion on the differences between text formatters and word processors, and their relative strengths and weaknesses.

A *word processor* is a program that to some extent simulates a typewriter — text is edited and formatted by one program. You type text at a computer terminal, and the word processor formats the text on the screen for you as you go. You usually get special effects like underlining and boldface by typing control indicators. The word processor usually displays these activated features using inverse video or special marks on the screen. The document is displayed on the terminal screen in the same format as it will appear on the printing device. The effects of this are often termed 'What You See Is What You Get' (usually called WYSIWYG and pronounced 'wizzi-wig'). Unfortunately, as has been pointed out, the problem with many WYSIWYG editors is that 'What You See Is *All* You Get'. In general, word processors cannot handle large documents. In principle, it is possible to write large manuals and even whole books with word processors, but the process gets painful for large manuscripts. Sometimes a change, such as deleting a sentence or inserting a new one, in the early part of a document can require that the whole document has to be reformatted. A change in the overall structure of the formatting requirements (for example, a changed indentation depth) will also mean that the whole document has to be reformatted. Word processors usually don't cope with automatic chapter and section numbering (of the kind you see in the Sun manuals), neither can they generate tables of contents and indices automatically. These tasks have to be done manually, and are a potential source of error. Word processors are eminently suitable for memos and letters, and can handle short documents. But large documents, or formatting documents for sophisticated devices like modern phototypesetters, requires a text formatter.

A *text formatter* such as `nroff` or `troff` does not in general perform any editing — its only job is reading text from a file and formatting that text for printing on some device. Entering the text into the file, and formatting the text from that file for printing are two separate and independent operations. You prepare your file of text using a text editor such as `vi` (described elsewhere in this manual). The file contains text to be formatted, interspersed with formatting instructions which control the layout of the final text. The text formatter reads this file of text, and obeys the formatting instructions contained in the file. The results of the formatting process is a finished document. The disadvantage of a text formatter is that you have to run them to find out what the final result will look like. Many people find the idea of embedded 'formatting commands' foreign, as they do the idea of two separate processes (an edit followed by a run of the formatter) to get the final document.

Notwithstanding all of the above, the UNIX system has had text formatting utilities since the very beginning, and many documents were written using the capabilities of `nroff` or `troff`.

### The Evolution of `nroff` and `troff`

One of the very first text formatting programs was called *runoff* and was a utility for the Compatible Time Sharing System (CTSS) at MIT in the early 1960's. *Runoff* was named for the way that people would say 'I'll just run off a document'.

When the UNIX system came to have a text formatter, the text formatter was called *roff*, because UNIX people like to call things by short and cryptic names. *Roff* was a simple program that was easy to work with as long as you were writing very small and simple documents for a line-printer. In some ways, *roff* is easier to use than `nroff` or `troff` because *roff* had built-in facilities such as being able to specify running headers and footers for a document with simple commands.

`nroff` stands for 'Newer *roff*'. `troff` is an adaptation of `nroff` to drive a phototypesetting machine. Although `troff` is supposed to mean 'typesetter *roff*', some people have formed the theory that `troff` actually stands for 'Times Romanoff' because of `troff`'s penchant for the Times Roman typeface.

`nroff` and `troff` are much more flexible (and much more complicated) programs — it's safe to say that they don't do a lot for you — for instance, you have to manage your own pagination, headers, and footers. The way that `nroff` and `troff` ease the burden is via facilities to define your own text formatting commands (macros), define strings, and store and manipulate numbers. Without these facilities, you would go mad (many people have — the author of this document among them). In addition, there are supporting packages for doing special effects such as mathematics and tabular layouts.

### Preprocessors and Postprocessors

Because troff or nroff are so hard to use 'in the raw', various tools have evolved to convert from human-oriented ways of specifying things into codes that troff or nroff can understand. Tools that do translations for troff or nroff before the fact are called *preprocessors.* There are also tools that hack over the output of nroff for different devices or for other requirements. Tools that do conversions of troff or nroff output after the fact are called *postprocessors.* Refer to the manual *Formatting Documents* for explanations of nroff and troff pre- and postprocessors.

## 1.2. troff, Typesetters, and Special-Purpose Formatters

*Please be sure to read this* : this section covers some aspects of troff that are generally glossed over in the traditional UNIX system manuals. troff was originally designed as a text formatter targeted to one specific machine — that machine was called a Graphics Systems Incorporated (GSI) C/A/T (Computer Assisted Typesetter). The C/A/T is a strange and wonderful device with strips of film mounted on a revolving drum, lenses, and light pipes. The C/A/T flashes character images on film which you then develop to produce page proofs for your book or manual or whatever. The C/A/T is almost extinct now except for some odd niches like Berkeley.

troff was written very much with the C/A/T in mind. The internal units of measurement that troff uses are C/A/T units, troff only understands four fonts at a time, and so on. Throughout this chapter, much of the terminology is based on troff's intimate relationship with the C/A/T.

## 1.3. Using the nroff and troff Text Formatters

To use nroff or troff you first prepare your file of text with nroff or troff requests embedded in the file to control the formatting actions. The remainder of this document discusses the formatting commands. Then you run the formatter at the command level like this:

```
hostname% nroff options files
```

or, of course:

```
hostname% troff options files
```

where *options* represents any of a number of option arguments and *files* represents the list of files containing the document to be formatted.

An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input. If no file names are given, input is taken from the standard input.

Options may appear in any order so long as they appear before the files. There are three parts to the list of options below: the first list of options are common to both nroff and troff; the second list of options are only applicable to nroff; the third list of options are only applicable to troff.

Each option is typed as a separate argument — for example,

```
hostname% nroff -o4,8-10 -T300S -ms file1 file2
```

formats pages 4, 8, 9, and 10 of a document contained in the files named *file1* and *file2*, specifies the output terminal as a DASI-300S, and invokes the **–msun** macro package.

### Options Common to `nroff` and `troff`

**-o***list*
: Print 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*
: Number first generated page *N*.

**-s***N*
: Stop 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*
: Adds the macro file `/usr/lib/tmac/tmac.`*name* before the input *files*.

**-r***aN*
: Register *a* (one-character) is set to *N*.

**-i** Read standard input after the input files are exhausted.

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

**-z** Suppress formatted output. The only output you get are messages from `.tm` (terminal message) requests, and from diagnostics.

### Options Applicable Only to `nroff`

**-h** Output tabs used during horizontal spacing to speed output as well as reduce byte count. Device tab settings assumed to be every 8 nominal character widths. Default settings of input (logical) tabs is also initialized to every 8 nominal character widths.

**-T***name*
: Specifies the name of the output terminal type. Currently-defined names are `37` for the (default) Model 37 Teletype®, `tn300` for the GE TermiNet 300 (or any terminal without half-line capabilities), `300S` for the DASI-300S, `300` for the DASI-300, and `450` for the DASI-450 (Diablo Hyterm).

**-e** Produce equally-spaced words in adjusted lines, using full terminal resolution.

### Options Applicable Only to troff

**-t** Direct output to the standard output instead of the phototypesetter.

**-a** Send a printable (ASCII) approximation of the results to the standard output.

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

## 1.4. General Explanation of troff and nroff Source Files

This section of the nroff and troff manual covers generic topics related to the format of the input file, how requests are formed, and how numeric parameters to requests are stated.

To use troff, you have to prepare not only the actual text you want printed, but some information that tells *how* you want it printed. For troff, the text and the formatting information are often intertwined. Most commands to troff are placed on a line separate from the text itself, beginning with a period (one command per line). For example:

```
Here is some text in the regular size characters,
but we want to make some of the text in a
.ps 14
larger size to emphasize something
```

changes the 'point size', that is, the size of the letters being printed, to '14 point' (one point is 1/72 inch) like this:

Here is some text in the regular size characters, but we want to make some of the text in a larger size to emphasize something

Occasionally, though, something special occurs in the middle of a line — to produce Area $= \pi r^2$ you have to type

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

(which we will explain shortly). The backslash character (\) introduces troff commands and special characters within a line of text.

To state the above more formally, an input file to be processed by troff or nroff consists of *text lines*, which are destined to be printed, interspersed with *control lines*, which set parameters or otherwise control subsequent processing. A control line is usually called a request.

A request begins with a *control character* — normally . (period) or ′ (apostrophe or acute accent) — followed by a one or two character name. A request is either:

*a basic request*
(also called a command) which is one of the many predefined things that nroff or troff can do. For example, .ll 6.5i is a basic request to set the line-length to 6.5 inches, and .in 5 is a basic request to indent the left margin by five en-spaces.

*a macro reference*

specifies substitution of a user-defined *macro* in place of the request. A *macro* is a predefined collection of basic requests and (possibly) other macros. For example, in the **–ms** macro package discussed elsewhere in this manual, `.LP` is a macro to start a new left-blocked paragraph.

The ´ (apostrophe or acute accent) control character suppresses the *break* function— 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 white space (spaces and/or tabs) for aesthetic reasons. Names must be followed by either space or newline. `nroff` or `troff` *ignores* control lines whose names are unrecognized.

Various special functions may be introduced anywhere in the input by means of an *escape* character, normally `\`. For example, the function `\n`*R* interpolates the contents of the *number register* whose name is *R* in place of the function. Here *R* is either a single character name in which case the escape sequence has the form `\n`*x*, or else *R* is a two-character name, in which case the escape sequence must have the form `\n(`*xx*. In general, there are many escape sequences whose one-character form is `\f`*x* and whose two-character form is `\f(`*xx*, where `f` is the function and *x* or *xx* is the name.

To print the escape character (usually backslash), use `\e` (backslash e).

## Backspacing

Unless in *copy mode*, the ASCII backspace character is replaced by a backward horizontal motion having the width of the space character. Underlining as a form of line-drawing is discussed in the section on *Arbitrary Motions and Drawing Lines and Characters*. A generalized overstriking function is also described in the above- mentioned section.

## Comments

Comments may be placed at the *end* of any line by prefacing them with `\"`. A comment line cannot be continued by placing a \ at the end of the line — see the discussion on continuation lines below.

A line beginning with `\"` appears as a blank line and behaves like a `.sp 1` request:

```
Here is a line of text.
\" Here is a comment on a line by itself.
Here is another line of text.
```

when we format the above lines we get this:

```
Here is a line of text.

Here is another line of text.
```

If you want a comment on a line by itself but you don't want it to appear as a blank line, type it as `.\"`:

```
Here is a line of text
.\"  and here is a comment on a line by itself
and here is another line of text
```

when we format the above lines we get this:

```
Here is a line of text
and here is another line of text
```

### Continuation Lines

An uncomfortably long input line that must stay one line (for example, a string definition, or unfilled 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 — see below. This provides a continuation line facility. The `\` at the end of the line is called a *concealed newline* in the jargon.

### Transparent Throughput

An input line beginning with a `\!` is read in *copy mode* and *transparently* output (without the initial `\!`); the text processor is otherwise unaware of the line's presence. This mechanism may be used to pass control information to a post-processor or to embed control lines in a macro created by a diversion. Refer to Chapter 10 for information describing diversions.

### Formatter and Device Resolution

`troff` internally uses 432 units/inch, corresponding to the phototypesetter which has a horizontal resolution of 1/432 inch and a vertical resolution of 1/144 inch. `nroff` internally uses 240 units/inch, corresponding to the least common multiple of the horizontal and vertical resolutions of various typewriter-like output devices. `troff` rounds horizontal/vertical numerical parameter input to the actual horizontal/vertical resolution of the Graphic Systems typesetter. `nroff` similarly rounds numerical input to the actual resolution of the output device indicated by the `-T` option (default Model 37 Teletype).

### Specifying Numerical Parameters

Many requests can have numerical arguments. Both `nroff` and `troff` accept numerical input in a variety of units. The general form of such input is

```
.xx nnnn units
```

where *.xx* is the request, *nnnn* is the number, and *units* is the "scale indicator."

Scale indicators are 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.

Table 1-1 *Scale Indicators for Numerical Input*

| *Scale Indicator* | *Meaning* | *Number of basic units* | |
|---|---|---|---|
| | | `troff` | `nroff` |
| `i` | Inch | 432 | 240 |
| `c` | Centimeter | 432×50/127 | 240×50/127 |
| `P` | Pica = 1/6 inch | 72 | 240/6 |
| `m` | Em = *S* points | 6×*S* | *C* |
| `n` | En = Em/2 | 3×*S* | *C, same as Em* |
| `p` | Point = 1/72 inch | 6 | 240/72 |
| `u` | Basic unit | 1 | 1 |
| `v` | Vertical line space | *V* | *V* |
| none | Default, see below | | |

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, *V*s for the vertically-oriented requests and functions, *p* for the vertical spacing request; and *u* for the number register and conditional requests. See Table 1-2 for a summary of the default scale indicators for the `troff` requests and functions that take scale indicators.

Table 1-2 *Default Scale Indicators for Certain* `troff` *Requests and Functions*

| *Request* | *Default Scaling Unit* | *Request* | *Default Scaling Unit* |
|---|---|---|---|
| `.ll` | ems | `.pl` | vertical units (Vs) |
| `.in` | " | `.wh` | " |
| `.ti` | " | `.ch` | " |
| `.ta` | " | `.dt` | " |
| `.lt` | " | `.sp` | " |
| `.po` | " | `.sv` | " |
| `.mc` | " | `.ne` | " |
| `\h` | " | `.rt` | " |
| `\l` | " | `\v` | " |
| `.nr` | machine units (u) | `\x` | " |
| `.if` | " | `\L` | " |
| `.ie` | " | `.vs` | picas (p) |

*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 form, but the parameter finally stored is rounded to an integer number of basic units.

The *absolute position* indicator | (the pipe character) may precede a number *N* to generate the absolute distance to the vertical or horizontal place *N*. For vertically-oriented requests and functions, | *N* becomes the absolute distance in basic units from the current vertical place on the page or in a *diversion* (see Chapter 10 for the section on diversions) to the vertical place *N*. For *all* other requests and functions, | *N* becomes the distance from the current horizontal place on the *input* line to the horizontal place *N*. For example,

```
.sp | 3.2c
```

will space *in the required direction* to 3.2 centimeters from the top of the page.

## Numerical Expressions

Wherever numerical input is expected, you can type an arithmetic expression. An expression involves parentheses and the arithmetic operators and logical operators shown in the table below:

Table 1-3 *Arithmetic Operators and Logical Operators for Expressions*

| *Arithmetic Operator* | *Meaning* |
|---|---|
| + | Addition |
| – | Subtraction |
| / | Division |
| * | Multiplication |
| % | Modulo |
| *Logical Operator* | *Meaning* |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| = or == | Equal to |
| & | and |
| : | or |

Except where controlled by parentheses, evaluation of expressions is left-to-right — there is no operator precedence.

In certain requests, an initial + or – is stripped and interpreted as an increment or decrement indicator respectively. In the presence of default scaling, the desired scale indicator must be attached to *every* number in an expression for which the desired and default scaling differ. For example, if the number register x contains 2 and the current point size is 10, then

```
.ll  (4.25i+\nxP+3)/2u
```

will set the line length to 1/2 the sum of 4.25 inches + 2 picas + 30 points.

## 1.5. 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 error 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 — in the case of `troff`, the standard output should always be redirected unless the `-a` option is in effect, because `troff`'s output is a strange binary format destined to drive a typesetter.

Various *error* conditions may occur during the operation of `nroff` and `troff`. Certain less serious errors having only local impact do not stop processing. 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 philosophy is to continue processing, if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing terminates, and an appropriate message is printed. Examples are the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful.

# 2

# Line Format

# 2

# Line Format

Perhaps the most important reason for using `troff` or `nroff` is to use its filling and adjusting capabilities. Here is what filling and adjusting mean:

*Filling* means that `troff` or `nroff` collects words from your input text lines and assembles the collected words into an output text line until some word doesn't fit. An attempt is then made to hyphenate the word in an effort to assemble a part of it into the output line. Filling continues until something happens to break the filling process, such as a blank line in the text, or one of the `troff` or `nroff` requests that break the line — things that break the filling process are discussed later on.

*Adjusting* means that once the line has been filled as full as possible, spaces between words on the output line are then increased to spread out the line to the current line-length minus any current indent. The paragraphs you have just been reading are both filled and adjusted. Justification implies filling — it makes no sense to adjust lines without also filling them.

In the absence of any other information, `troff`'s or `nroff`'s standard behavior is to fill lines and adjust for straight left and right margins, so it is quite possible to create a neatly formatted document which only contains lines of text and no formatting requests. Given this as a starting point, the simplest document of all contains nothing but blocks of text separated by blank lines — `troff` or `nroff` will fill and justify those blocks of text into paragraphs for you. To get further control over the layout of text, you have to use requests and functions embedded in the text, and that is the subject of this entire paper on using `troff`.

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 by separating them with the *unpaddable space* character '`\ `' (backslash-space) — also called a 'hard blank' in other systems. The adjusted word spacings are uniform in `troff` and the minimum interword spacing can be controlled with the `.ss` (space size) request. In `nroff`, interword spaces are normally nonuniform because of quantization to character-size spaces, but the `-e` command line option requests uniform spacing to the full resolution of the output device. Multiple inter-word space characters found in the input are retained, except for trailing spaces.

Filling and adjusting and hyphenation can all be prevented or controlled by requests that are discussed later in this part of the manual.

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.

A text input line that happens to begin with a control character can be made to not look like a control line by prefacing it with the non-printing, zero-width filler character `\&`. Still another way is to specify output translation of some convenient character into the control character using the `.tr` (translate) request — see the relevant section.

The *text length* on the last line output is available in the `.n` number register, and text baseline position on the page for this line is in the `nl` number register. The text baseline high-water mark on the current page is in the `.h` number register.

## 2.1. Controlling Line Breaks

When filling is turned on, words of text are taken from input lines and placed on output lines to make the output lines as long as they can be without overflowing the line length, until something happens to break the filling process. When a break occurs, the current output line is printed just as it is, and a new output line is started for the following input text. There are various things that cause a break to occur:

Table 2-1 *Constructs that Break the Filling Process*

| *Construct* | *Explanation* |
|---|---|
| *Blank line(s)* | If your input text contains any completely blank lines, troff or nroff assumes you mean them. So it prints the current output line, then your blank lines, then starts the following text on a new line. |
| *Spaces* | at the beginning of a line are significant. If there are spaces at the start of a line, troff or nroff assumes you know what you are doing and that you really want spaces there. Obviously, to achieve this, the current output line must be printed and a new line begun. Avoid using tabs for this purpose, since they do not cause a break. |
| *A* .br request | A .br request (break) request can be used to make sure that the following text is started on a new line. |
| troff *or* nroff *requests* | Some troff or nroff requests cause a break in the filling process. However, there is an alternate format of these requests which does not cause a break. That is the format where the initial period character (.) in the request is replaced by the apostrophe or single quote character ('). The list of requests that cause a break appears in the table below this one. |
| *A* \p *Function* | When filling is in effect, the in-line \p function 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. |
| *End of file* | Filling stops when the end of the input file is reached. |

Breaks caused by blank lines or spaces at the beginning of a line enable you to take advantage of the filling and justification features provided by troff or nroff without having to use any troff or nroff requests in your text.

As mentioned in the table above in the item entitled "troff or nroff requests," there are some requests that cause a break when they are encountered. The list of requests that break lines is short and natural:

Table 2-2 *Formatter Requests that Cause a Line Break*

| *Command* | *Explanation* |
|---|---|
| `.bp` | Begin a new page |
| `.br` | Break the current output line |
| `.ce` | Center line(s) |
| `.fi` | Start filling text lines |
| `.nf` | Stop filling text lines |
| `.sp` | Space vertically |
| `.in` | Indent the left margin |
| `.ti` | Temporary indent the left margin for the next line only |

No other requests break lines, regardless of whether you use a `.` or a `'` as the control character. If you really *do* need a break, add a `.br` (break) request at the point at the appropriate place, as described below.

### `.br` — Break Lines

The `.br` (break) request breaks the current output line and stops filling that line. Any new output will start on a new line.

| | ***Summary of the*** **`.br`** ***Request*** |
|---|---|
| *Mnemonic:* | break |
| *Form of Request:* | `.br` |
| *Initial Value:* | Not Applicable |
| *If No Argument:* | cause break |
| *Explanation:* | Stop filling the line currently being collected and output the line without adjustment. Text lines beginning with space characters and empty text lines (blank lines) also cause a break. |

### Continuation Lines and Interrupted Text

The copying of an input line in *nofill* (non-fill) mode (see below) 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.

## 2.2. Justifying Text and Filling Lines

### .ad — Specify Adjusting Styles

To change the style of text justification, use the .ad (adjust) request to specify one of the four different methods for adjusting text:

Table 2-3 *Adjusting Styles for Filled Text*

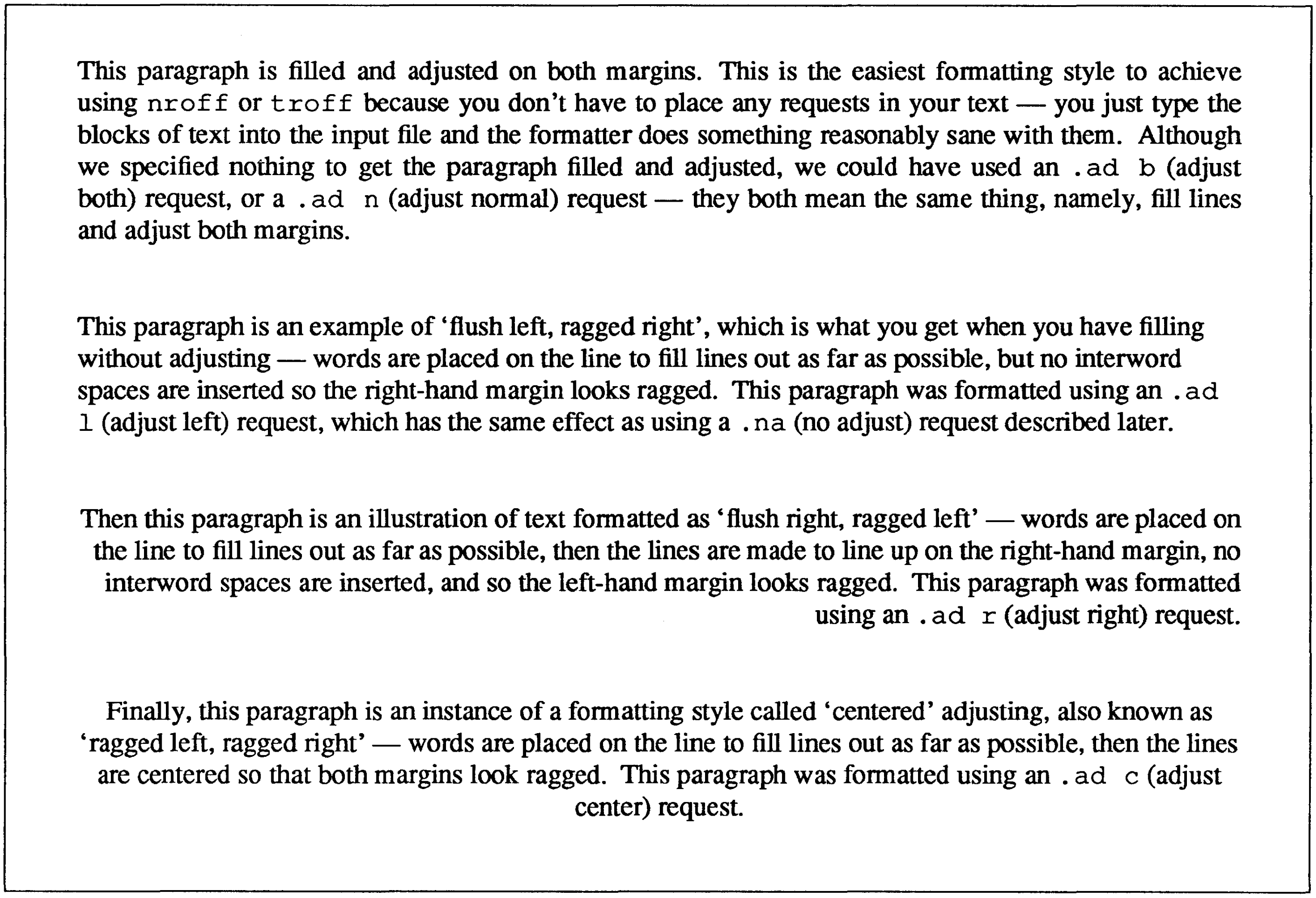
| *Adjusting Indicator* | *Adjusting Style* | *Description* |
|---|---|---|
| `.ad l` | Left | Produces flush-left, ragged-right output, which is the same as filling with no adjustment. |
| `.ad r` | Right | Produces flush-right, ragged-left output. |
| `.ad c` | Center | Centers each output line, giving both left and right ragged margins. |
| `.ad b`<br>`.ad n` | Both<br>Normal | Justifies both left and right margins. |
| `.ad` | Reset | Resumes adjusting lines in the last mode requested. |

It makes no sense to try to adjust lines when they are not being filled, so if filling is off when a .ad request is seen, the adjusting is deferred until filling is turned on again.

| ***Summary of the* .ad *Request*** | |
|---|---|
| *Mnemonic:* | adjust |
| *Form of Request:* | .ad *c* |
| *Initial Value:* | .ad b — that is, adjust both margins. |
| *If No Argument:* | Adjust in the last specified adjusting mode. |
| *Explanation:* | Adjust lines — if fill mode is off, adjustment is be deferred until fill mode is back on. If the type indicator *c* is present, the adjustment type is changed as shown in Table 2-3. |
| *Notes:* | E (see Table A-2) |

The current adjustment indicator *c* can be obtained from the .j number register.

The following figure illustrates the different appearances of filled and justified text.



This paragraph is filled and adjusted on both margins. This is the easiest formatting style to achieve using `nroff` or `troff` because you don't have to place any requests in your text — you just type the blocks of text into the input file and the formatter does something reasonably sane with them. Although we specified nothing to get the paragraph filled and adjusted, we could have used an `.ad b` (adjust both) request, or a `.ad n` (adjust normal) request — they both mean the same thing, namely, fill lines and adjust both margins.

This paragraph is an example of 'flush left, ragged right', which is what you get when you have filling without adjusting — words are placed on the line to fill lines out as far as possible, but no interword spaces are inserted so the right-hand margin looks ragged. This paragraph was formatted using an `.ad l` (adjust left) request, which has the same effect as using a `.na` (no adjust) request described later.

Then this paragraph is an illustration of text formatted as 'flush right, ragged left' — words are placed on the line to fill lines out as far as possible, then the lines are made to line up on the right-hand margin, no interword spaces are inserted, and so the left-hand margin looks ragged. This paragraph was formatted using an `.ad r` (adjust right) request.

Finally, this paragraph is an instance of a formatting style called 'centered' adjusting, also known as 'ragged left, ragged right' — words are placed on the line to fill lines out as far as possible, then the lines are centered so that both margins look ragged. This paragraph was formatted using an `.ad c` (adjust center) request.

Figure 2-1 *Filling and Adjusting Styles*

### `.na` — No Adjusting

If you don't specify otherwise, `troff` or `nroff` justifies your text so that both left and right margins are straight. This can be changed if necessary — one way, as we showed above, is to use the `.ad l` request to get left adjusting only so that the left margin is straight and the right margin is ragged. Another way to achieve this same effect is to use the `.na` (no adjust) request. Output lines are still filled, providing that filling hasn't also been turned off — see the `.nf` (no fill) and `.fi` (fill) requests below. If filling is still on, `troff` or `nroff` produces flush left, ragged right output. To turn adjusting back on (return to the previous state), use the `.ad` request.

| | ***Summary of the* `.na` *Request*** |
|---|---|
| *Mnemonic:* | no adjust |
| *Form of Request:* | `.na` |
| *Initial Value:* | Adjusting is on by default |
| *If No Argument:* | adjusting is turned off |
| *Explanation:* | Turn off adjustment — the right margin will be ragged. The adjustment type for the `.ad` request is not changed. Output lines are still *filled* if fill mode is on. To turn adjusting back on (return to the previous state), use the `.ad` request. |
| *Notes:* | E (see Table A-2) |

### `.nf` and `.fi` — Turn Filling Off and On

The `.nf` (no fill) request turns off filling. Lines in the result are neither filled nor adjusted. The output text appears exactly as it was typed in, complete with any extra spaces and blank lines you might type — this is often called 'as-is text', or 'verbatim'. No filling is mainly used for showing examples, especially in computer books where you want to show examples of program source code.

You should be aware that traditional typesetting people have trouble with the concept of no filling, because their typesetting systems are geared up to fill and adjust text all the time. When you ask for stuff to be printed exactly the way you typed it, they have problems, especially when you want blank lines left in the unfilled text exactly where you put them. In the world of typography, things that don't fit into the Procrustean mold of filled text are often called 'displays' and have to be handled specially.

The `.fi` (fill) request turns on filling. If adjusting has not been turned off by a `.na` request, output lines are also adjusted in the prevailing mode set by any previous `.ad` request.

| | ***Summary of the* `.fi` *Request*** |
|---|---|
| *Mnemonic:* | fill |
| *Form of Request:* | `.fi` |
| *Initial Value:* | Filling is on by default |
| *If No Argument:* | filling is turned on |
| *Explanation:* | Fill subsequent output lines. The number register `.u` is 1 in fill mode and 0 in nofill mode. |
| *Notes:* | E,B (see Table A-2) |

***Summary of the* `.nf` *Request***

| | |
|---|---|
| *Mnemonic:* | no fill |
| *Form of Request:* | `.nf` |
| *Initial Value:* | Filling is on by default |
| *If No Argument:* | filling is turned off |
| *Explanation:* | Subsequent output lines are *neither* filled *nor* adjusted. Input text lines are copied directly to output lines *without regard* for the current line length. The number register `.u` is 1 in fill mode and 0 in nofill mode. |
| *Notes:* | E,B (see Table A-2) |

## 2.3. Hyphenation

When `troff` or `nroff` fills lines, it takes each word in turn from the input text line, and puts the word on the output text line, until it finds a word that will not fit on the output line. At this point, `troff` or `nroff` tries to hyphenate the word. If possible, the first part of the hyphenated word is put on the output line followed by a –, and the remainder of the word is put on the next line. We should emphasize that, although the examples show text that is both filled and justified, it is during filling that `troff` or `nroff` hyphenates words, not adjusting.

If you have words in your input text containing hyphens (such as jack-in-the-box, or co-worker), `troff` or `nroff` will, if necessary, split these words over two lines, even if hyphenation is turned off.

### `.nh` and `.hy` — Control Hyphenation

Normally, when you invoke `troff` or `nroff`, hyphenation is turned on, but you can change this. The `.nh` (no hyphenation) request turns off automatic hyphenation. When hyphenation is turned off, the only words that are split over more than one line are those that already contain hyphens. Hyphenation can be turned on again with the `.hy` (hyphenate) request.

You can give `.hy` an argument to restrict the amount of hyphenation that `troff` or `nroff` does. The argument is numeric. The request `.hy 2` stops `troff` or `nroff` from hyphenating the last word on a page. `.hy 4` instructs `troff` or `nroff` not to split the last two characters from a word; so, for example, 'repeated' will never be hyphenated 'repeat-ed'. `.hy 8` requests the same thing for the first two characters of a word; so, for example, 'repeated' will not be hyphenated 're-peated'.

The values of the arguments are additive: `.hy 12` makes sure that words like 'repeated' will never be hyphenated either as 'repeat-ed' or as 're-peated'. `.hy 14` calls up all three restrictions on hyphenation.

A `.hy 1` request is the same as the simple `.hy` request — it turns on hyphenation everywhere. Finally, a `.hy 0` request is the same as the `.nh` request — it turns off automatic hyphenation altogether.

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

### *Summary of the* `.nh` *Request*

| | |
|---|---|
| *Mnemonic:* | no hyphenation |
| *Form of Request:* | `.nh` |
| *Initial Value:* | Hyphenation is on by default |
| *If No Argument:* | hyphenation is turned off |
| *Explanation:* | Turn automatic hyphenation off. |
| *Notes:* | E (see Table A-2) |

### *Summary of the* `.hy` *Request*

| | |
|---|---|
| *Mnemonic:* | hyphenation |
| *Form of Request:* | `.hy` *N* |
| *Initial Value:* | Hyphenation is on by default in mode 1. |
| *If No Argument:* | $N=1$. |
| *Explanation:* | Turn automatic hyphenation on for $N \geq 1$, or off for $N=0$. If $n=1$, all words are subject to hyphenation. If $N=2$, do not hyphenate *last* lines (ones that cause a trap). If $N=4$, do not hyphenate the *last* two characters of a word. If $N=8$, do not hyphenate the *first* two characters of a word. These values are additive — that is, $N=14$ invokes all three restrictions. Note: odd values of $N$ (except 1) don't make sense. |
| *Notes:* | E (see Table A-2) |

### `.hw` — Specify Hyphenation Word List

If there are words that you want `troff` or `nroff` to hyphenate in some special way, you can specify them with the `.hw` (hyphenate words) request. This request tells `troff` or `nroff` that you have special cases it should know about, for example:

```
.hw pre-empt ant-eater
```

Now, if either of the words 'preempt' or 'anteater' need to be hyphenated, they will appear as specified in the `.hw` request, regardless of what `troff` or `nroff` 's usual hyphenation rules would do. If you use the `.hw` request, be aware that there is a limit of about 128 characters in total, for the list of special words.

***Summary of the* `.hw` *Request***

| | |
|---|---|
| *Mnemonic:* | hyphenate word |
| *Form of Request:* | `.hw` *word1* ... |
| *Initial Value:* | None |
| *If No Argument:* | Ignored |
| *Explanation:* | Specify hyphenation points in words with embedded minus signs. Versions of a word with terminal *s* are implied — that is, *dig–it* implies *dig–its*. This list is examined initially *and* after each suffix stripping. The space available is small — about 128 characters. |

### `.hc` — Specify Hyphenation Character

A *hyphenation indicator* character may be embedded in a word to specify desired hyphenation points, or may precede the word to suppress hyphenation. For example, hyphenation looks particularly disruptive if it occurs in titles. So, if you had a long title like:

*Input and Output Conventions and Character Translations*,

you could shorten it, or you could insert the hyphenation character just before the first character of each of the long words at the end of the title. The input might look like this:

```
.H C "Input and Output Conventions and \%Character \%Translations"
```

(If you are using a reasonable line length, you don't need to worry about hyphenation occurring earlier in the title in this example.)

Here is an example of using the hyphenation character to specify acceptable hyphenation points within a word. The word "workstation" is often mis-hyphenated because of the collection of consonants at the end of "work" and the beginning of "station". So, your input might look like this:

```
work\%station
```

***Summary of the*** `.hc` ***Request***

| | |
|---|---|
| *Mnemonic:* | hyphenation character |
| *Form of Request:* | `.hc` *c* |
| *Initial Value:* | `\%` |
| *If No Argument:* | `\%` |
| *Explanation:* | Set hyphenation indicator character to *c* or to the default `\%`. The indicator does not appear in the output. |
| *Notes:* | E (see Table A-2) |

## 2.4. `.ce` — Center Lines of Text

When we described "Filling and Adjusting," we showed how the text produced by `nroff` or `troff` could be centered by using the `.ad c` request. Setting text adjustment for centering is a fairly unusual way of getting centered text, because the text is being filled at the same time. The more usual use for centering is to have unfilled lines that are centered — that is, each line that you type is centered within the output line. You get lines centered via the `.ce` (center) request, which centers lines of text.

If you just use a `.ce` request without an argument, `troff` or `nroff` centers the *next* line of text:

```
.ce
```

centers the following line of text, whereas:

```
.ce 5
```

centers the following five lines of text. Filling is temporarily turned off when lines are centered, so each line in the input appears as a line in the output, centered between the left and right margins. For centering purposes, the left margin includes both the page offset (see later) and any indentation (also see later) that may be in effect.

An argument of zero to the `.ce` request simply stops any centering that might be in progress. So, if you don't want to count how many lines you want centered, you can ask for some large number of lines to be centered, then follow the last of the lines with a `.ce 0` request:

```
.ce 100
     .
     .
     .
lines of text to be centered
     .
     .
     .
.ce 0
```

The '100' in the example above could be any large number that you think is bigger than the number of lines to center.

Note that the argument to the `.ce` request only applies to following text lines in the input. Lines containing `nroff` or `troff` requests are not counted.

| | ***Summary of the* `.ce` *Request*** |
|---|---|
| *Mnemonic:* | center |
| *Form of Request:* | `.ce` *N* |
| *Initial Value:* | Centering is off by default. |
| *If No Argument:* | *N*=1 |
| *Explanation:* | Center the next *N* input text lines within the current line (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 is left adjusted. |
| *Notes:* | E,B (see Table A-2) |

## 2.5. `.ul` and `.cu` — Underline or Emphasize Text

There are times when you want to lend *emphasis* to a word in a piece of text. The normal way to do this is to place the word or piece of text in *italics* if you have an italic font, or underline the word if you don't have an italic font. The `.ul` (underline) request underlines alphanumeric characters in `nroff`, and prints those characters in the italic font in `troff`. As with the `.ce` request, a `.ul` request with no argument underlines a single line of text, so:

```
.ul
following line of text
```

simply underlines the following line of text. Unlike `.ce`, though, `.ul` does not turn filling off. A numeric argument to the `.ul` request specifies the number of text lines you want underlined, so:

```
.ul 3
```

underlines the next three lines of text. As with centering, an argument of zero `.ul 0` cancels the underlining process.

| | ***Summary of the* `.ul` *Request*** |
|---|---|
| *Mnemonic:* | underline |
| *Form of Request:* | `.ul` *N* |
| *Initial Value:* | Underlining is off by default. |
| *If No Argument:* | $N$=1 |
| *Explanation:* | Underline in `nroff` (italicize in `troff`) the next *N* input text lines. Actually, switch to *underline* font, saving the current font for later restoration; *other* font changes within the span of a `.ul` will take effect, but the restoration will undo the last change. Output generated by a `.tl` request *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. |
| *Notes:* | E (see Table A-2) |

Another form of underlining is called up with the `.cu` request, and asks for continuous underlining. This is the same as the `.ul` request, except that *all* characters are underlined. Again, if you are using `troff` the characters are printed in the italic font instead of underlined. There is a way to get characters underlined in `troff`, and this technique is explained later in this manual.

As with `.ce`, only lines of text to be underlined are counted in the number given to the underline request. `nroff` or `troff` requests interspersed with the text lines are not counted.

| | ***Summary of the* `.cu` *Request*** |
|---|---|
| *Mnemonic:* | continuously underline |
| *Form of Request:* | `.cu` *N* |
| *Initial Value:* | Underlining is off by default. |
| *If No Argument:* | $N$=1 |
| *Explanation:* | A variant of `.ul` that underlines *every* character in `nroff`. Identical to `.ul` in `troff`. |
| *Notes:* | E (see Table A-2) |

## 2.6. `.uf` — Underline Font

`nroff` automatically underlines characters in the *underline* font, specifiable with a `.uf` (underline font) request. The underline font is normally Times Italic and is mounted on font position 2. In addition to the `.ft` (font) request and the `\f`*F*, the underline font may be selected by the `.ul` (underline) request and the `.cu` (continuous underline) request. Underlining is restricted to an output-device-dependent subset of *reasonable* characters.

| ***Summary of the `.uf` Request*** | |
|---|---|
| *Mnemonic:* | underline font |
| *Form of Request:* | `.uf` *F* |
| *Initial Value:* | Italic |
| *If No Argument:* | Italic |
| *Explanation:* | Set underline font to *F*. In `nroff`, *F* may *not* be on position 1 (initially Times Roman). |

# 3

# Page Layout

# 3

# Page Layout

Now we get into the subject of altering the physical dimensions of the layout of text on a page. There are two major parts to page control, and they can be roughly divided into controlling the *horizontal* aspects of lines, and controlling the *vertical* aspects of the page dimensions.

*Horizontal page control* Deals with subjects such as the location of the left margin, the location of the right margin (the length of the line), and indentation of lines.

*Vertical page control* Deals with the physical length of the page, when pages get started, and whether there's enough room on the current page for a block of text. Page numbering is also covered in this area.
These topics are covered in this section. We deal first with horizontal page control, then with the vertical aspects of page control.

We should explain how `troff` thinks of a page. The next page contains a diagram of a page of text, and here we explain what some of the terms mean:

*Page Offset* is the distance from the physical edge of the paper to the place where all text begins. In normal-world terms, this distance is called the 'left margin'. Normally you only set the page-offset at the very start of a formatting job and you never change it again.

*Line Length* is the distance from the left margin (or page-offset) to the right edge of the text. The line-length is *relative* to the page-offset. In some respects, 'line-length' is a bit of a misnomer, because once you have set the page-offset at the start of the document (and assuming you never change it), the line-length really nails down the position of the *right margin* and has little to do with the length of the line.

*Indent* is where the left edge of your text starts. Normally the indent is zero, so that the edge of the text is where the page-offset is, but you can change the indent so that the text starts somewhere else. Note that the line-length is not affected by the indent — that is, indenting the text doesn't change the position of the right margin.

*Page Length* is the distance from the extreme top of the page to the extreme bottom of the page, that is, the page length is the physical length of the paper.

The following figure is a diagram of a page of text with the relevant parts pointed out. This diagram is a scale-model of an 8.5 × 11-inch sheet of paper, so while the numbers quoted in the text below are expressed in 'real' units, the actual dimensions are scaled.

Figure 3-1 *Layout of a Page*

left header center header right header

This paragraph has the page-offset set to give a left margin of approximately one inch (scaled). The line-length is set to 6.5 inches (scaled). This means there is a one-inch (scaled) left margin and a one-inch (scaled) right margin. The indent is set to zero so that the current left margin is at the same place as the page-offset.

> This paragraph has the page-offset and the line-length the same as the last paragraph, but we've used a `.in +0.5i` request to indent the left margin by half an inch — the current left margin is now page-offset + indent. Note that the position of the right margin remains the same as in the previous paragraph — only the left margin moved, so the effective length of the lines is shorter.

This paragraph now has the left margin back to the original position because we inserted a `.in -0.5i` request before it.

> This paragraph could have the left margin moved, not by indenting, but by changing the page-offset via a `.po +0.5i` request. Now *all* text would be moved to the left, and because the line-length hasn't changed, the right margin would move as well. The example can't show this because page offset is measured from the margin, and because this example is in a box, changing the page offset within the box is meaningless.

This is the regular old paragraph where the first line is indented and the rest of the text in the paragraph is flushed to the left margin. The first line was indented via a `.ti +0.25i` request to give a temporary indent of the first line.

- This paragraph is an example of an 'item' or 'bulleted' or 'hanging' paragraph, where the left margin is moved to the right, and the 'bullet' or 'tag' is moved back to the old left margin. This effect was achieved via a `.in +0.25i` request to move the left margin rightward, and then the 'bullet' was preceded by a `.ti -0.25i` request to get a temporary indent to the old position of the left margin.

Finally, note that tab stops are relative to the current left margin as we show here with a couple of blocks of text with different indents. Note that the positions of the tab stops are shown with exclamation point (`!`) characters:
! ! ! ! ! !
You can see by the line of `!` marks above where the tab stops are.

> Now we have another block of text here but with the indent moved over a half-inch. As you can see by the line of `!` marks below, the tab stops have moved with the left margin:
> ! ! ! ! ! !

left footer center footer right footer

## 3.1. Margins and Indentations

As we said above, the positions of the left-hand and right-hand margins are controlled via the page-offset and the line-length. After that, any movements of the left-hand margin are controlled via indent and temporary indent requests. These topics are discussed in the following subsections.

### `.po` — Set Page Offset

The usable page width on the Graphic Systems phototypesetter is about 7.54 inches, beginning about 1/27 inch from the left edge of the 8 inch wide, continuous roll paper. The physical limitations on `nroff` output are output-device dependent.

The page-offset is the distance from the extreme left-hand edge of the paper to the left margin of your text. When you use 'standard' 8.5×11-inch paper, it is customary to have the left and right margins be one inch each, so that the physical length of the printed lines are 6.5 inches — or you'd say that the measure was 39 picas if you're a typographer and can't handle inches.

In general, you only set the page-offset once in the course of formatting a document. Setting the page-offset determines the position of the physical left margin for the text, and then you (almost) never change the page-offset again — all indentation is done via `.in` (indent) requests and `.ti` (temporary indent) requests. We talk about these requests later in this part of the manual.

The position of the physical right margin for the text is determined by the line-length relative to the page-offset. The `.ll` (line length) request is discussed below.

***Summary of the* `.po` *Request***

| | |
|---|---|
| *Mnemonic:* | page offset |
| *Form of Request:* | `.po` ±*N* |
| *Initial Value:* | 0 in `nroff`, 26/27 inch in `troff`. |
| *If No Argument:* | Previous value |
| *Explanation:* | Set the current *left margin* to ±*N*. In `troff` the initial value is 26/27 inch, which provides about one 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. In `nroff` the initial page-offset is zero. |
| *Notes:* | v (see Table A-2) |

The current page-offset is available in the `.o` register.

### `.ll` — Set Line Length

`troff` gives you full control over the length of the printed lines. By the way, typographers don't use terms like 'line-length', they use the word 'measure' to mean the length of a line. They always measure vertical distances in 'picas'.

Nevertheless, to set the line-length in `troff`, use the `.ll` (line length) request, as in

```
.ll 6i
```

As with the `.sp` request, the actual length can be specified in several ways — inches are probably the most intuitive unless you live in one of the very few places in the world where they don't use inches.

The maximum line-length provided by the typesetter is 7.5 inches, by the way. To use the full width, you have to reset the default physical left margin ('page-offset'), which is normally slightly less than one inch from the left edge of the paper. This is done by the `.po` (page offset) request discussed above.

```
.po 0
```

sets the offset as far to the left as it will go.

Note that the line-length *includes* indent space but *not* page-offset space. The line-length minus the indent is the basis for centering with the `.ce` request. The effect of the `.ll` request 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 is available in the `.l` number register. The length of three-part titles produced by a `.tl` request (see Chapter 7, *Titles and Page Numbering*) is independent of the line-length set by the `.ll` request — the length of a three-part title is set by the `.lt` request.

***Summary of the*** `.ll` ***Request***

| | |
|---|---|
| *Mnemonic:* | line length |
| *Form of Request:* | `.ll` ±*N* |
| *Initial Value:* | 6.5 inches |
| *If No Argument:* | Previous value |
| *Explanation:* | Set the line-length to *N* where *N* is the value of the line length, or an increment or decrement for the line-length. In `troff` the maximum (line-length)+(page-offset) is about 7.54 inches. |
| *Notes:* | E, m (see Table A-2) |

### `.in` — Set Indent

Given that you've got your page-offset and line-length correctly set for a document to establish the position of the left and right margins, you now make all other movements of the left margin via the `.in` (indent) request discussed here, and via the `.ti` (temporary indent) request described below.

The `.in` (indent) request indents the left margin by some specified amount from the page-offset. This means that all the following text will be indented by the specified amount until you do something to change the indent. To get only the first line of a paragraph indented, you don't use the `.in` request, but you use the

.ti (temporary indent) request described below.

As an example, a common text structure in books and magazines is the 'quotation' — a paragraph that is indented both on the right and the left of the line. A quotation is used for precisely that purpose, namely to set some text off from the rest of the copy. We can achieve such a paragraph by using the .in request to move the left margin in, and the .ll request to move the right margin leftward:

```
.in +0.5i
.ll -0.5i
I was to learn later in life that we tend to meet any new
situation by reorganizing; and a wonderful method
it can be for creating the illusion of progress
while producing confusion, inefficiency, and demoralization.
.ll +0.5i
.in -0.5i
```

When you format the above construct you get a block that looks like this:

> I was to learn later in life that we tend to meet any new situation
> by reorganizing; and a wonderful method it can be for creating
> the illusion of progress while producing confusion, inefficiency,
> and demoralization.[2]

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 +2.0i makes lines two inches longer, whereas .ll 2.0i makes them two inches long:

```
.ll 2.0i
I was to learn later in life that we tend to meet any new
situation by reorganizing; and a wonderful method
it can be for creating the illusion of progress
while producing confusion, inefficiency, and demoralization.
```

> I was to learn later in life that
> we tend to meet any new situa-
> tion by reorganizing; and a
> wonderful method it can be for
> creating the illusion of progress
> while producing confusion,
> inefficiency, and demoraliza-
> tion.

With .in, .ll, and .po, the previous value is used if no argument is specified. So, in the above example, the lines:

[2] *Petronius Arbiter*, A.D. 60.

```
.li +0.5i
.in -0.5i
```

could have been

```
.li
.in
```

and would have had the same effect.

Note that the line-length *includes* indent space but *not* page-offset space. The line-length minus the indent is the basis for centering with the `.ce` request. The effect of the `.in` request 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 indent is available in the `.i` number register.

| | ***Summary of the*** `.in` ***Request*** |
|---|---|
| *Mnemonic:* | indent |
| *Form of Request:* | `.in` ±*N* |
| *Initial Value:* | 0 |
| *If No Argument:* | Previous value |
| *Explanation:* | Set the indent to ±*N* where *N* is the value of the indent, or an increment or decrement on the current value of the indent. The `.in` request causes a *break*. |
| *Notes:* | E, m (see Table A-2) |

### `.ti` — Temporarily Indent One Line

The `.ti` (temporary indent) request indents the *next* text line by a specified amount.

A common application for `.ti` is where the first line of a paragraph must be indented just like the one you're reading now. You get such a construct with a sequence like:

```
.ti 3
A common application for . . .
          .
          .
          .
```

and when the paragraph is formatted, the first line of the paragraph is indented by three specified units just like this one. Three of what? The default unit for the `.ti` request, as for most horizontally-oriented requests — `.ll` (line length), `.in` (indent), and `.po` (page offset) — is ems. An *em* is roughly the width of the letter 'm' in the current point size. Thus, an em is always *proportional* to the point size you are using. An em in size *p* is the number of *p* points in the width of an 'm'. Here's an em followed by an em dash in several point sizes to show why this is a *proportional* unit of measure. You wouldn't want a 20-point dash if you are printing the rest of a document in 12-point text. Here's 12-point text:

m
|—|

Here's 16-point text:

m
|—|

And here's 20-point text:

m
|—|

Thus a temporary indent of `.ti 3` in the current point size results in an indent of three m's width or |mmm|.

Although inches are usually clearer than ems to people who don't set type for a living, ems have a place: they 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 as scale factors directly, as in `.ti 2.5m`.

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

```
.ti -0.3i
```

moves the next line back three tenths of an inch. A common text structure found in documents is 'itemized lists' where the paragraphs are indented but are set off by 'bullets' or some such. Item lists are often called 'hanging paragraphs' because the first line with the item on it 'hangs' to the left. For example, you could type the following series of lines like this (we've deliberately shortened the length of the line to illustrate the effects):

```
.ll 4.0i                       shorten lines for this example
.in +0.2i                      indent left margin by a fifth inch
.ta +0.2i                      set a tab for the hanging indent
.ce                            center a line of title
Indent Control Requests
.ti -0.2i                      move left margin back temporarily
\(bu tab the \fL\&.po\fP request sets the
page-offset to the desired amount thereby making
sure the left margin is correct.
.ti -0.2i                      move left margin back temporarily
\(bu tab the \fL\&.in\fP request sets the
indent from the left margin for all following text.
.ti -0.2i                      move left margin back temporarily
\(bu tab the \fL\&.ti\fP request sets the indent for
the following line of text only, thus providing for
fancy paragraph effects.
```

We had to play some tricks with tabs as well to get everything lined up, but that won't affect the main point of the discussion. The *tab* markers in the lines above show where there's a tab character, and the `\(bu` sequence at the start of the lines gets you a bullet (•) like that — we'll show the special character sequences later in this manual. When you format the text as shown in the example above, you get this effect:

Indent Control Requests

- the `.po` request sets the page-offset to the desired amount thereby making sure the left margin is correct.
- the `.in` request sets the indent from the left margin for all following text.
- the `.ti` request sets the indent for the following line of text only, thus providing for fancy paragraph effects.

Remember that the line-length *includes* indent space but *not* page-offset space. The effect of a `.ti` request 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 indent is available in the `.i` register.

### *Summary of the* `.ti` *Request*

| | |
|---|---|
| *Mnemonic:* | temporary indent |
| *Form of Request:* | `.ti` ±*N* |
| *Initial Value:* | 0 |
| *If No Argument:* | Ignored |
| *Explanation:* | Indent the *next* output text line a distance ±*N* with respect to the current indent. The resulting total indent may not be negative. The current indent is not changed. The `.ti` request causes a *break*. |
| *Notes:* | E, m (see Table A-2) |

## 3.2. Page Lengths, Page Breaks, and Conditional Page Breaks

Neither `nroff` nor `troff` provide any facilities for top and bottom margins on a page, nor for any kind of page numbering at all. The **–ms** macro package described in a previous section of this manual sets things up so that reasonable pagination with top and bottom margins and page numbers is done automatically.

If you want top and bottom margins when using raw `troff` or `nroff`, you have to do some tricky stuff. The tricky stuff is done via *traps* and *macros*. The trap tells `troff` or `nroff` *when* to do some processing for the margins (for example, you might set a trap to start the bottom margin 0.75 inches from the bottom of the page), and the *macro* defines *what* to do when the trap is sprung. It is conventional to 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 *non-diverted* text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition.

In the following tables, references to the *current diversion* mean that the mechanism being described works during both ordinary and diverted output (the former considered as the top diversion level). Refer to Chapter 10 for more information on diversions.

### `.pl` — Set Page Length

Just as the `.po`, `.ll`, `.in`, and `.ti` requests changed the horizontal aspects of the page, the `.pl` (page length) request determines the physical length of the page. In general you won't need to use the `.pl` request because the standard setting is right for all but the most esoteric purposes.

| | ***Summary of the* `.pl` *Request*** |
|---|---|
| *Mnemonic:* | page length |
| *Form of Request:* | `.pl` ±*N* |
| *Initial Value:* | 11 inches |
| *If No Argument:* | 11 inches |
| *Explanation:* | Set page length to ±*N*. The internal limitation is about 75 inches in `troff` and about 136 inches in `nroff`. The current page length is available in the `.p` number register. |
| *Notes:* | v (see Table A-2) |

### `.bp` — Start a New Page

This request causes a break and skips to a new page.

***Summary of the*** `.bp` ***Request***

| | |
|---|---|
| *Mnemonic:* | begin page |
| *Form of Request:* | `.bp` ±*N* |
| *Initial Value:* | *N*=1 |
| *If No Argument:* | Increment current page number by 1. |
| *Explanation:* | Eject the current page and start a new page. If ±*N* is given, the new page number will be ±*N*. Also see the `.ns` (no space) request. The `.bp` request causes a *break*. |
| *Notes:* | v (see Table A-2) |

### `.pn` — Set Page Number

***Summary of the*** `.pn` ***Request***

| | |
|---|---|
| *Mnemonic:* | page number |
| *Form of Request:* | `.pn` ±*N* |
| *Initial Value:* | *N*=1 |
| *If No Argument:* | Ignored |
| *Explanation:* | The next page (when it occurs) will have the page number ±*N*. A `.pn` request must occur before the initial pseudo-page transition to affect the page number of the first page. The current page number is in the `%` register. |

### `.ne` — Specify Space Needed

In some applications you need to make sure that a few lines of text all appear together on the same page. There are several ways to achieve this ranging from simple to complicated. One of the simplest ways is to use the `.ne` (need) vertical space request:

```
.ne 3                specify we need at least three lines
some
lines
of
text
to
be
kept
on the
same page
```

The arrangement of the `.ne` request specifies that if there are many lines of text in (say) a paragraph, at least three of the lines will appear together on the same page, otherwise a new page will be started. The object of this exercise is to avoid what typographers call 'orphans' — that is, the first line of a paragraph appearing all alone and lonely on the bottom of a page, while the rest of the paragraph appears on the next page. This is generally considered to be somewhat ugly and should be avoided if possible. By itself, `troff` is too stupid to recognize the existence of orphans (indeed of any text constructs at all), but the facilities are there to avoid these situations. In general, macro packages such as the **–ms** macro package discussed elsewhere have 'begin paragraph' macros such as `.PP` which take care of controlling orphans.

| | ***Summary of the*** `.ne` ***Request*** |
|---|---|
| *Mnemonic:* | need |
| *Form of Request:* | `.ne` *N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | 1V |
| *Explanation:* | 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. |
| *Notes:* | v (see Table A-2) |

## 3.3. Multi-Column Page Layout by Marking and Returning

It is possible to achieve multi-column output in `troff` or `nroff` via the `.mk` (mark) and `.rt` (return) requests. Other useful special effects can also be obtained using these requests, but one of the common uses is to do multi-column output. Basically, the `.mk` request marks the current vertical position on the page (you can place the result of the mark in a register). You do a column's worth of output, then when you get to the end of the page, instead of starting the next page, you return (via the `.rt` request) to the marked position, set up a new indent and line-length, and crank out another column.

### `.mk` — Mark Current Vertical Position

| | ***Summary of the*** `.mk` ***Request*** |
|---|---|
| *Mnemonic:* | mark |
| *Form of Request:* | `.mk` *R* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | *R* is an internal register |
| *Explanation:* | Mark the *current* vertical place in an internal register (both associated with the current diversion level), or in register *R*, if given. See the `.rt` request. |

### `.rt` — Return to Marked Vertical Position

***Summary of the*** **`.rt`** ***Request***

| | |
|---|---|
| *Mnemonic:* | return |
| *Form of Request:* | `.rt` ±*N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | return to place marked by a previous `.mk` request. |
| *Explanation:* | Return *upward only* to a marked vertical place in the current diversion. If ±*N* (with respect to the current place) 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 (refer to the chapter *Line Spacing and Character Sizes*) may be used in all cases instead of `.rt` by spacing to the absolute place stored in a explicit register; for example, using the sequence `.mk` *R* `. . . .sp ~ \n`*R*`u`. |

# 4

# Line Spacing and Character Sizes

# 4

# Line Spacing and Character Sizes

## 4.1. .sp — Space Vertically

You get extra vertical space with the .sp (space) request. A simple

```
.sp
```

request with no argument gives you one extra blank line (one .vs, whatever that has been set to). Typically, that's more or less than you want, so .sp can be followed by information about how much space you want —

```
.sp 2i
```

means 'two inches of vertical space'.

```
.sp 2p
```

means 'two points of vertical space'; and

```
.sp 2
```

means 'two vertical spaces' — two of whatever .vs is set to (this can also be made explicit with .sp 2v); troff also understands decimal fractions in most places, so

```
.sp 1.5i
```

is a space of 1.5 inches. These same scale factors can be used after the .vs request to define line spacing, and in fact after most requests that deal with physical dimensions.

It should be noted that all size numbers are converted internally to 'machine units', which are 1/432 inch (1/6 point). For most purposes, this is enough resolution that you don't have to worry about the accuracy of the representation. The situation is not quite so good vertically, where resolution is 1/144 inch (1/2 point).

| | |
|---|---|
| | ***Summary of the*** `.sp` ***Request*** |
| *Mnemonic:* | space |
| *Form of Request:* | `.sp` *N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | *N*=1V |
| *Explanation:* | 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 the no-space mode is on, no spacing occurs (see `.ns`, and `.rs` below). |
| *Notes:* | B, v (see Table A-2) |

### 4.2. `.ps` — Change the Size of the Type

In `troff`, you can change the physical size of the characters that are printed on the page. The `.ps` (point size) request 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. `troff` and the machine it was originally designed for understand 15 point sizes, listed below.

6 point: Pack my box with five dozen liquor jugs.
7 point: Pack my box with five dozen liquor jugs.
8 point: Pack my box with five dozen liquor jugs.
9 point: Pack my box with five dozen liquor jugs.
10 point: Pack my box with five dozen liquor jugs.
11 point: Pack my box with five dozen liquor jugs.
12 point: Pack my box with five dozen liquor jugs.
14 point: Pack my box with five dozen liquor jugs.
16 point: Pack my box with five dozen liquor jugs.
18 point: Pack my box with five dozen liquor jugs.
20 point: Pack my box with five dozen liquor jugs.
22 point: Pack my box with five dozen liquor jugs.
24 point: Pack my box with five dozen liquor jugs.
28 point: Pack my box with five dozen liquor
36 point: Pack my box with five doz

If the number after a `.ps` request is not one of these legal sizes, it is rounded up to the next valid value, with a maximum of 36. If no number follows `.ps`, `troff` reverts to the previous size, whatever it was. `troff` begins with point size 10, which is usually fine. This document is in 11-point.

The point size can also be changed in the middle of a line or even a word with an in-line size change sequence. In general, text which is in ALL CAPITALS in the middle of a sentence tends to loom large over the rest of the text and so it is customary to drop the point size of the capitals so that it looks like ALL CAPITALS instead. You use the `\s` (for size) sequence to state what the point size should be. You can state the size explicitly as in this line here:

```
The \s8POWER\s0 of a \s8SUN\s0
```

to produce the output line like:

The POWER of a SUN

As above, `\s` should be followed by a legal point size, except that `\s0` makes the size revert to its previous value (before you just changed it).

Note that because there are a fixed number of point sizes that the system knows about, the sequence `\s96` gets you a nine-point 6 instead of 96-point type like you wanted, whereas the sequence `\s180` gets you an 18-point 0 instead of 180-point type.

Stating the point size in absolute terms as above is not always a good idea — what you really want is for the changed size to be relative to the surrounding text, so that if your document is in 11-point type like this one, you'd really like the bigger (or smaller stuff) to be a couple of points different without your having to know explicitly what the actual size is. So in this case, you can use a relative size-change sequence of the form `\s+` *n* to raise the point size, and `\s−` *n* to lower the point size. The number *n* is restricted to a single digit. So we can rework our previous example from above like this:

```
The \s-2POWER\s+2 of a \s-2SUN\s+2
```

to produce the output line like:

```
The POWER of a SUN
```

Relative size changes have the advantage that the size difference is independent of the starting size of the document. Of course this stuff only works really well (in typography terms) when the changes in size aren't too violently out of whack with the point size — a change of two points in 36-point type doesn't have quite the same impact as it does for 12-point type — there is a question of the weight of the type, but by the time you get to that stuff you'll be much more knowledgeable about typography.

The current size is available in the `.s` number register. `nroff` ignores type size control.

| ***Summary of the* `.ps` *Request*** | |
|---|---|
| *Mnemonic:* | point size |
| *Form of Request:* | `.ps` ±*N* |
| *Initial Value:* | 10 points |
| *If No Argument:* | Previous value |
| *Explanation:* | Set point-size to ±*N*. Alternatively embed `\s`*N* or `\s`±*N*. Any positive size value may be requested; if invalid, the next larger valid size will result, with a maximum of 36. The sequence<br>`.ps` +*N*<br>`.ps` *N*<br>works the same as<br>`.ps` +*N*<br>`.ps` –*N*<br>because the previous requested value is also remembered. Ignored in `nroff`. |
| *Notes:* | E (see Table A-2) |

## 4.3. `.vs` — Change Vertical Distance Between Lines

The other parameter that determines what the type looks like is the spacing between lines, which is set independently of the point size. Vertical spacing is measured from the bottom of one line to the bottom of the next. The bottom of the text on a line is often called the *baseline*. The vertical spacing is often called *leading* (pronounced 'led-ing') and comes from the days when text was produced with lead slugs instead of electronic widgets like laser printers.

You control vertical spacing with the `.vs` (vertical spacing) request. For running text, it is usually best to set the vertical spacing about 20% bigger than the character size. For example, so far in this document, we have used 11-point type with a vertical line-spacing of 13 points between baselines. Typographers call this '11 on 13', so when you hear some one say that a book is set in '11 on 13', you know that it's 11-point type with 13-point vertical spacing.

So, somewhere at the start of this document, the macro package that formats this document for us had requests like:

```
.ps 11p
.vs 13p
```

Had we set the point size and the vertical spacing like this:

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

the running text would look like this. After a few lines, you will agree it looks a little cramped. The right vertical spacing is partly a matter of taste, depending on how much text you want to squeeze into a given space, and partly a matter of traditional printing style. By default, `troff` 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 6 on 7, which is even smaller. It packs a lot more words per line, but you can go blind trying to read it.**

When used without arguments, both `.ps` and `.vs` revert to the previous size and vertical spacing respectively.

The vertical spacing *(V)* between the base-lines 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. This document is set in 11-point type with a 13-point vertical spacing. The current *V* is available in the `.v` number register.

***Summary of the* `.vs` *Request***

| | |
|---|---|
| *Mnemonic:* | vertical spacing |
| *Form of Request:* | `.vs` *N* |
| *Initial Value:* | 1/6 inch in `nroff`, 12 points in `troff`. |
| *If No Argument:* | Previous value |
| *Explanation:* | Set vertical base-line spacing size *V*. Transient *extra* vertical space available with `\x`'*N*' (see section on `\x` Function). |
| *Notes:* | E, p (see Table A-2) |

## 4.4. `.ls` — Change Line Spacing

Multiple-*V* line separation (for instance, double spacing) can be requested with the `.ls` (line spacing) request.

| ***Summary of the*** `.ls` ***Request*** | |
|---|---|
| *Mnemonic:* | line spacing |
| *Form of Request:* | `.ls` *N* |
| *Initial Value:* | *N*=1 |
| *If No Argument:* | Previous value |
| *Explanation:* | Set *line* spacing to ±*N*. *N*–1 *V*s *(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. |
| *Notes:* | E (see Table A-2) |

### 4.5. `\x` Function — Get Extra Line-Space

If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before and/or after it, the *extra-line-space* function `\x`'*N*' can be embedded in or attached to that word. In this and other functions having a pair of delimiters around their parameter (here ' ), 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 used post-line extra line-space is available in the `.a` register.

### 4.6. `.sv` — Save Block of Vertical Space

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

| ***Summary of the*** `.sv` ***Request*** | |
|---|---|
| *Mnemonic:* | save space |
| *Form of Request:* | `.sv` *N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | *N*=1V |
| *Explanation:* | 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 (see the `.os` request). Subsequent `.sv` requests will overwrite any still-remembered *N*. |
| *Notes:* | v (see Table A-2) |

## 4.7. `.os` — Output Saved Vertical Space

***Summary of the*** `.os` ***Request***

| | |
|---|---|
| *Mnemonic:* | output saved space |
| *Form of Request:* | `.os` |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Output saved vertical space |
| *Explanation:* | 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. |

## 4.8. `.ns` — Set No Space Mode

***Summary of the*** `.ns` ***Request***

| | |
|---|---|
| *Mnemonic:* | no-space mode |
| *Form of Request:* | `.ns` |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Turn on no-space mode |
| *Explanation:* | Turn on no-space mode — 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`. |
| *Notes:* | D (see Table A-2) |

## 4.9. `.rs` — Restore Space Mode

***Summary of the*** `.rs` ***Request***

| | |
|---|---|
| *Mnemonic:* | restore space mode |
| *Form of Request:* | `.rs` |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Turn off no-space mode |
| *Explanation:* | Restore spacing — turn off no-space mode. |
| *Notes:* | D (see Table A-2) |

## 4.10. `.ss` — Set Size of Space Character

| | ***Summary of the*** `.ss` ***Request*** |
|---|---|
| *Mnemonic:* | space-character size |
| *Form of Request:* | `.ss` *N* |
| *Initial Value:* | 12/36 em |
| *If No Argument:* | Ignored |
| *Explanation:* | Set space-character size to *N*/36 ems. This size is the minimum word spacing in adjusted text. Ignored in `nroff`. |
| *Notes:* | E (see Table A-2) |

## 4.11. `.cs` — Set Constant-Width Characters

| | ***Summary of the*** `.cs` ***Request*** |
|---|---|
| *Mnemonic:* | constant spacing |
| *Form of Request:* | `.cs` *F N M* |
| *Initial Value:* | Off |
| *If No Argument:* | Ignored |
| *Explanation:* | Constant character space (width) mode is set on for font *F* (if mounted); the width of every character is taken as *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 still or again in effect when the characters are physically printed. Ignored in `nroff`. |
| *Notes:* | P (see Table A-2) |

5

# Fonts and Special Characters

# 5

# Fonts and Special Characters

`troff` and the typesetter allow four different fonts at any one time. Normally three fonts (Times Roman, italic and bold) and one collection of special characters are permanently mounted.

abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJ KLMNOPQRSTUVWXYZ
*abcdefghijklmnopqrstuvwxyz 0123456789*
*ABCDEFGHIJKLMNOPQRSTUVWXYZ*
**abcdefghijklmnopqrstuvwxyz 0123456789**
**ABCDEFGHIJKLMNOPQRSTUVWXYZ**

The Greek, mathematical symbols, and miscellany of the special font are listed in Appendix B, *Font and Character Examples.*

`troff` prints in Roman unless told otherwise. To switch into bold, use the `.ft` (font) request:

```
.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`.

## 5.1. `.ft` — Set Font

| | ***Summary of the*** `.ft` ***Request*** |
|---|---|
| *Mnemonic:* | font |
| *Form of Request:* | `.ft` *F* |
| *Initial Value:* | Roman |
| *If No Argument:* | Previous Font |
| *Explanation:* | Change font to *F*. Alternatively, embed `\f`*F*. The font name `P` is reserved to mean the previous font. |
| *Notes:* | E (see Table A-2) |

The 'underline' request

```
.ul
```

makes the next input line print in italics. `.ul` can be followed by a count to indicate that more than one line is to be italicized. Refer to Chapter 2 for a more detailed description of the `.ul` request.

Fonts can also be changed within a line or word with the in-line request `\f`:

**bold***face* text

is produced by the input

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

If you want to do this so the previous font, whatever it was, is left undisturbed, insert extra in-line `\fP` commands, like this:

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

Because only the immediately previous font is remembered, you have to restore the previous font after each change or you lose it. The same is true of `.ps` and `.vs` when used without an argument.

There are other fonts available besides the standard set, although you can still use only four at any given time. The `.fp` (font position) request 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` requests should appear at the beginning of your document if you do not use the standard fonts.

It is possible to make a document relatively independent of the actual fonts used to print it by using font numbers instead of names; for example, `\f3` and `.ft 3` mean 'whatever font is mounted at position 3', and thus work for any setting. Normal settings are Roman font (R) on font position 1, italic (I) on position 2, bold (B) on position 3, and special (S) on position 4 — the mnemonic 'R I B S' might help you remember.

## 5.2. `.fp` — Set Font Position

| | ***Summary of the*** `.fp` ***Request*** |
|---|---|
| *Mnemonic:* | font position |
| *Form of Request:* | `.fp` *N F* |
| *Initial Value:* | R, I, B, S |
| *If No Argument:* | Ignored |
| *Explanation:* | 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 that can be mounted on a numbered quadrant of a wheel. The default mounting sequence assumed by `troff` is R, I, B, and S on positions 1, 2, 3 and 4. Any `.fp` request specifying a font on some position must precede `.fz` requests relating to that position. |

## 5.3. `.fz` — Force Font Size

| | ***Summary of the*** `.fz` ***Request*** |
|---|---|
| *Mnemonic:* | font size |
| *Form of Request:* | `.fz` *S F N* |
| *Initial Value:* | None |
| *If No Argument:* | None |
| *Explanation:* | Forces font *F* or *S* for special characters to be in size *N*. A `.fz 3 -2` causes implicit \s–2 every time font 3 is entered, and a matching \s+2 when left. Same for special font characters that are used during *F*. Use *S* to handle special characters during *F*. `.fz 3 -3` or `.fz S 3 -0` causes automatic reduction of font 3 by 3 points while special characters are not affected. Any `.fp` request specifying a font on some position must precede `.fz` requests relating to that position. |

There is also a way to get 'synthetic' bold fonts by overstriking letters with a slight offset. Look at the `.bd` request.

## 5.4. .bd — Artificial Boldface

| | ***Summary of the* .bd *Request*** |
|---|---|
| *Mnemonic:* | bold |
| *Form of Request:* | .bd *F N* |
| *Initial Value:* | Off |
| *If No Argument:* | No Emboldening |
| *Explanation:* | Artificially embolden characters in font *F* 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 still or again in effect when the characters are physically printed. Ignored in nroff. |
| *Form of Request:* | .bd S *F N* |
| *Explanation:* | Embolden characters in the special font whenever the current font is *F*. The mode must be still or again in effect when the characters are physically printed. |
| *Notes:* | P (see Table A-2) |

Special characters have four-character names beginning with \(, and they may be inserted anywhere. For example,

¼ + ½ = ¾

is produced by

```
\(14 + \(12 = \(34
```

In particular, Greek letters are all of the form \(**x*, where *x* represents an upper- or lower-case Roman letter reminiscent of the Greek. Thus to get

```
Σ(α×β) → ∞
```

in raw troff we have to type

```
\(*S(\(*a\(mu\(*b) \(-> \(if
```

That line is unscrambled as follows:

| *Escape Sequence* | *Character Printed* | *Description* |
|---|---|---|
| `\(*S` | Σ | *Upper-case Sigma or Sum* |
| ( | ( | |
| `\(*a` | α | *lower-case alpha* |
| `\(mu` | × | *multiplication sign or signum* |
| `\(*b` | β | *lower-case beta* |
| ) | ) | |
| `\(->` | → | *tends toward* |
| `\(if` | ∞ | *infinity* |

A complete list of these special names occurs in Appendix B, *Font and Character Examples.*

In `eqn`, explained in the chapter "Formatting Mathematics with `eqn`" in *Formatting Documents*, you can achieve the same effect with the input

```
SIGMA ( alpha times beta ) -> inf
```

which is less concise (31 keystrokes instead of 27!), but clearer to the uninitiated.

Notice that each four-character name is a single character as far as `troff` is concerned. For example, the translate request

```
.tr \(mi\(em
```

is perfectly clear, meaning

```
.tr – —
```

that is, to translate – (minus sign) into — (em-dash).

Some characters are automatically translated into others: grave ` and acute ´ accents (apostrophes) become open and close single quotes ‘ ’; the combination of ‘‘...’’ is generally preferable to the double quotes "...". Similarly a typed minus sign becomes a hyphen -. To print an explicit – sign, use `\-`. To get a backslash printed, use `\e`.

## 5.5. Character Set

The `troff` character set consists of the Graphics Systems Commercial II character set plus a Special Mathematical Font character set — each having 102 characters. These character sets are shown in Appendix B, *Font and Character Examples.* 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 also explained in Appendix B. The three ASCII exceptions are mapped as follows:

Table 5-1 *Exceptions to the Standard ASCII Character Mapping*

| *ASCII Input* | | *Printed by* troff | |
|---|---|---|---|
| *Character* | *Name* | *Character* | *Name* |
| ´ | acute accent | ’ | close quote |
| ` | grave accent | ‘ | open quote |
| − | minus | - | hyphen |

The characters ´, `, and − may be input by \´, \`, and \− respectively or by their names found in Appendix B. The ASCII characters @, #, ", ´, `, <, >, \, {, }, ~, ^, and _ exist *only* on the Special Font and are printed as a one-em space if that font is not mounted.

nroff understands the entire troff character set, but can in general print only ASCII characters, additional characters as may be available on the output device, such characters as may be constructed by overstriking or other combination, and those that can reasonably be mapped into other printable characters. The exact behavior is determined by a driving table prepared for each device. The characters ´, `, and _ print as themselves.

## 5.6. Fonts

The default mounted fonts are Times Roman (R), Times Italic (I), Times Bold (B), and the Special Mathematical Font (S) on physical typesetter positions 1, 2, 3, and 4 respectively. These fonts and others are used in this document. The *current* font, initially Roman, may be changed (among the mounted fonts) by use of the .ft request, or by embedding at any desired point either \f*x*, \f(*xx*, or \f*N* where *x* and *xx* are the name of a mounted font and *N* is a numerical font position. It is *not* necessary to change to the Special font; characters on that font are automatically handled. A request for a named but not-mounted font is *ignored.* troff can be informed that any particular font is mounted by use of the .fp request. The list of known fonts is installation-dependent. In the subsequent discussion of font-related requests, *F* represents either a one- or two-character font name or the numerical font position, 1 through 4. The current font is available (as numerical position) in the read-only number register .f.

nroff understands font control and normally underlines italic characters.

## 5.7. .lg — Control Ligatures

A ligature is a special way of joining two characters together as one. Way back in the days before Gutenberg, scribes would have a variety of special forms to choose from to make lines come out all the same length on a manuscript. Some of these forms are still with us today.

Five ligatures are available in the current troff character set — **fi, fl, ff, ffi,** and **ffl**. They may be input (even in nroff) by \(fi, \(fl, \(ff, \(Fi, and \(Fl respectively.

The ligature mode is normally on in troff, and *automatically* invokes ligatures during input.

If you want other ligatures like the æ, œ, Æ, andŒ ligatures, you have to make them up yourself — `troff` doesn't know about them. See Chapter 12 the section on "Arbitrary Horizontal Motion" (the `\h` function) for some examples on constructing these ligatures.

| | ***Summary of the* `.lg` *Request*** |
|---|---|
| *Mnemonic:* | ligature |
| *Form of Request:* | `.lg` *N* |
| *Initial Value:* | Off in `nroff`, on in `troff`. |
| *If No Argument:* | on |
| *Explanation:* | Turn Ligature mode on if *N* is absent or non-zero. Turn ligature mode 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 file names, and in *copy mode*. No effect in `nroff`. |

# 6

# Tabs, Leaders, and Fields

# 6

# Tabs, Leaders, and Fields

There are several ways to get stuff lined up in columns, and to achieve other effects such as horizontal motion and repeated strings of characters. The three related topics we discuss in this section are *tabs*, *leaders*, and *fields*.

*tabs* behave just like the tab stops on a typewriter.

*leaders* are for generating repeated strings of characters.

*fields* are a general mechanism for helping to line stuff up into columns.

This part of the document concentrates on the 'easy' parts, so to speak. Later sections of this document contain discussions on the facilities for drawing lines and for producing arbitrary motions on the page.

## 6.1. `.ta` — Set Tabs

Tabs (the ASCII horizontal tab character) can be used to produce output in columns, or to set the horizontal position of output. Typically tabs are used only in unfilled text. Tab stops are set by default every half inch from the current indent (in `troff`) and every 0.8 inch from the current indent (in `nroff`), but can be changed by the `.ta` (tab) request. In the example below, we set tab stops every one-and-a-half inches and set some text in columns based on those tab stops. We place a line of exclamation marks (`!`) above and below the text to show where the tabs stops are in the output page:

```
.ta 1.5i 3.0i 4.5i 6.0i                 set tabs
! tab ! tab ! tab ! tab !               show where tabs are with ! character
word-one tab word-two tab word-three tab word-four tab word-five
! tab ! tab ! tab ! tab !
```

When we format the above example, we get this output:

```
!               !               !               !               !
word-one        word-two        word-three      word-four       word-five
!               !               !               !               !
```

### Setting Relative Tab Stops

The tab stops set in the example above are in terms of *absolute* position on the line. You could also set tabs *relative* to previous tabs stops by preceding the tab stop number with a + sign, and get exactly the same result:

```
.ta 1.5i +1.5i +1.5i +1.5i                set tabs
! tab ! tab ! tab ! tab !                          show where tabs are with ! character
word-one tab word-two tab word-three tab word-four tab word-five
! tab ! tab ! tab ! tab !
```

### Right-Adjusted Tab Stops

In the standard case as shown in the above examples, the tab stops are left-adjusted (as on a typewriter). You can also make the tab stops right-adjusting for doing things like lining up columns of numbers. When you right-adjust a tab stop, the action of placing a tab before the field places the material *behind* the tab stop on the output line. Here's an example of some input with both alphabetic and numeric items:

```
.nf
.ta 2.0iR
July tab 5
August tab 9
September tab 15
October tab 60
November tab 85
December tab 126
.fi
```

Notice the `.ta` request — it has the letter `R` on the end to indicate that this is a right-adjusted tab. When we format that table, we get this result:

```
July          5
August        9
September    15
October      60
November     85
December    126
```

Notice how the numbers in the second column line up.

### Centered Tab Stops

Finally you can make a *centered* tab stop, so that things get centered between the tabs. We can use the centering tabs to put a title on our table from above:

```
.nf
.ta 2.0iC
Month tab Shipments
.ta 2.0iR
July tab 5
August tab 9
September tab 15
October tab 60
November tab 85
December tab 126
.fi
```

and when we format this table now, we get this result:

```
Month            Shipments
July                 5
August               9
September           15
October             60
November            85
December           126
```

Notice that the column headings are centered over the data in the table.

If you have a complex table, instead of using `troff` or `nroff` directly, use the `tbl` program described in the chapter "Formatting Tables with `tbl`" in *Formatting Documents.* A good example of where `tbl` does more work for you is when numerically-aligned items have decimal points in them — it is really hard to do this using the raw `troff` or `nroff` capabilities.

### `.tc` — Change Tab Replacement Character

A tab inserts blank spaces between the item that came before and after it. You can change this by filling up tabbed-over space with some other character. Set the 'tab replacement character' with the `.tc` (tab character) request:

```
.ta 2.5i 4.5i
.tc _
Name tab Age tab
```

This produces

Name ____________________________ Age ______________________

There is a more general mechanism for drawing lines, described in the sections "Drawing Vertical Lines" and "Drawing Horizontal Lines" in the chapter "Arbitrary Motions and Drawing Lines and Characters."

To reset the tab replacement character to a space, use the `.tc` request with no argument. Lines can also be drawn with the in-line `\l` command, described in the chapter "Arbitrary Motions and Drawing Lines and Characters."

| | ***Summary of the* `.tc` *Request*** |
|---|---|
| *Mnemonic:* | tab character |
| *Form of Request:* | `.tc` *c* |
| *Initial Value:* | space |
| *If No Argument:* | Removed |
| *Explanation:* | The tab repetition character becomes *c*, or is removed, specifying motion. |
| *Notes:* | E (see Table A-2) |

### Summary of Tabs

The table below is a summary of the types of tab stops. 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 was found) to the next tab stop.

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

*W* is the width of *next-string*.

Table 6-1 *Types of Tab Stops*

| *Tab letter* | *Tab type* | *Length of motion or repeated characters* | *Location of next-string* |
|---|---|---|---|
| *blank* | Left | *D* | Following *D* |
| R | Right | *D–W* | Right adjusted within *D* |
| C | Centered | *D–W*/2 | Centered on right end of *D* |

### *Summary of the* `.ta` *Request*

| | |
|---|---|
| *Mnemonic:* | tab |
| *Form of Request:* | `.ta` *Nt*... |
| *Initial Value:* | 0.8 inches in `nroff`, 0.5 inches in `troff`. |
| *If No Argument:* | Ignored |
| *Explanation:* | Set tab stops and types — *N* is the tab stop value and *t* is the type. `troff` tab stops are preset every 0.5 inches; `nroff` tab stops are preset every 0.8 inches. *t*=`R` means right-adjusting tabs, *t*=`C` means centering tabs, and if *t* is absent, the tabs are left-adjusting tab stops. Stop values in the list of tab stops are separated by spaces, and a value preceded by + is treated as an increment to the previous stop value. |
| *Notes:* | E, m (see Table A-2) |

## 6.2. Leaders — Repeated Runs of Characters

Leaders are repeated runs of the same character between tab stops. Leaders are most often used to hang two separated pieces of text together. A common application is in tables of contents. If you look at the contents for this manual you will see that the chapter and section titles (on the left of the line) are separated from the page number (on the right end of the line) by a row of dots. In fact here is a short example to illustrate what the leaders look like:

```
                         Contents

2.0 Blunt Uses of Clubs ...........................................  13
  2.1 Social Clubs ................................................  16
  2.2 Arthritic Clubs .............................................  18
  2.3 Golf Clubs ..................................................  25
  2.4 Two-by-Four Clubs ...........................................  29
```

The dots are called *leaders*, because they ‘lead’ your eye from one thing to the other. It is not nearly so easy to read stuff like that if the leaders aren't there:

```
                         Contents

2.0 Blunt Uses of Clubs                                              13
  2.1 Social Clubs                                                   16
  2.2 Arthritic Clubs                                                18
  2.3 Golf Clubs                                                     25
  2.4 Two-by-Four Clubs                                              29
```

The leader character is normally a period, but it can in fact be any character you like — some people prefer dots and some people prefer a solid line:

```
                         Contents

2.0 Blunt Uses of Clubs ______________________________  13
  2.1 Social Clubs ___________________________________  16
  2.2 Arthritic Clubs ________________________________  18
  2.3 Golf Clubs _____________________________________  25
  2.4 Two-by-Four Clubs ______________________________  29
```

A leader is very similar to a tab, but you get the repeated characters by typing an in-line `\a` sequence instead of a tab or a `\t` sequence. The `\a` sequence is a control-A character or an ASCII SOH (start of heading) character and is hereafter known as the *leader* character for the purposes of this discussion. When the leader character is encountered in text it generates a string of repeated characters. The length of the repeated string of characters is governed by internal tab stops specified just as for ordinary tabs as discussed in the section on tabs above. The major difference between tabs and leaders is that tabs generate *motion* and leaders generate a *string of periods*. Let's look at a fragment of the text that generated the examples above:

```
.DS
.ta 5.0i-5nR 5.0iR
2.0  Blunt Uses of Clubs  \a\t13"
  2.1  Social Clubs  \a\t16"
  2.2  Arthritic Clubs  \a\t18"
  2.3  Golf Clubs  \a\t25"
  2.4  Two-by-Four Clubs  \a\t29"
.DE
```

What we're trying to get here are lines of text with the section numbers and the titles, followed by a string of leader characters, followed by some space and then the page number at the right-hand end of the line. Tables of contents tend to look better with shorter line lengths, so we set our first tab to five inches minus five en-spaces to leave a gap at the end of the leader. The second tab is set to a right-adjusting tab at five inches. Each line of the table now contains the text to appear on the left end, followed by a couple of spaces, followed by the `\a` sequence to indicated the leader, followed by the `\t` sequence to indicate the tab, and finally followed by the page number. The result of formatting all that stuff is:

```
2.0 Blunt Uses of Clubs ..............................  13
  2.1 Social Clubs ...................................  16
  2.2 Arthritic Clubs ................................  18
  2.3 Golf Clubs .....................................  25
  2.4 Two-by-Four Clubs ..............................  29
```

## `.lc` — Change the Leader Character

Just as you could use the `.tc` request to change the character that gets generated with tabs, you can use the `.lc` (leader character) request to specify the character that is generated by a leader. The standard leader character is the period. We can show this by taking our last fragment and placing a `.lc` request before it to change the leader character to an underline:

```
.DS
.lc _                                   set leader character
.ta 5.0i-5nR 5.0iR                      set tabs
2.0  Blunt Uses of Clubs  \a\t13"
  2.1  Social Clubs  \a\t16"
  2.2  Arthritic Clubs  \a\t18"
  2.3  Golf Clubs  \a\t25"
  2.4  Two-by-Four Clubs  \a\t29"
.DE
```

Then when we format the thing, it looks like this:

```
2.0 Blunt Uses of Clubs _____________________________   13
  2.1 Social Clubs _________________________________    16
  2.2 Arthritic Clubs ______________________________    18
  2.3 Golf Clubs ___________________________________    25
  2.4 Two-by-Four Clubs ____________________________    29
```

Whereas the length of generated motion for a tab can be negative, the length of a repeated character string cannot be. Repeated character strings contain an integer number of characters, and any residual distance is added before the leaders as space. 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 non-interpreted tab and leader respectively, and are equivalent to actual tabs and leaders in *copy mode.*

### ***Summary of the*** `.lc` ***Request***

| | |
|---|---|
| *Mnemonic:* | leader character |
| *Form of Request:* | `.lc` *c* |
| *Initial Value:* | . |
| *If No Argument:* | Removed — successive `\a`s act like tabs |
| *Explanation:* | The leader repetition character becomes *c*, or is removed. Successive leader requests (`\a`s) act like tabs. |
| *Notes:* | E (see Table A-2) |

### 6.3. .fc — Set Field Characters

A field is a more general mechanism for laying out material between tab stops. Hardly anyone ever needs to use fields, but the tbl preprocessor uses them for placing tabular material on the page. This section is a very short discussion on how to use fields. In general, when you want to lay out tabular material you should use tbl to do the job for you. Fields are a way of reducing the number of tab stops you have to set, and also have troff or nroff do some automatic work in parceling out padding space for you.

A field lives between the current position on the input line and the next tab stop. The start and end of the field are indicated by a field delimiter character. troff or nroff places the field on the line and pads out any excess space with spaces. You indicate where the padding actually goes by placing padding indicator characters at various places in the field. You set the field delimiter character and the padding indicator character with the .fc (field characters) request. In the absence of any other information, troff or nroff has the field mechanism turned off entirely. The .fc request looks like:

```
.fc dp
```

where *d* is the field delimiter character and *p* is the padding indicator character. If you do not specify any character for a padding indicator, the space character is the default. However, this means that you could not have spaces within the field, so you normally specify the padding indicator as something other than a space.

So let's start with a very simple example of a single field and see what we get. Here is the input:

```
.ta 3.0i                  set a single tab at three inches
.fc # @                   set field delimiter character to # and
                          set padding indicator character to @
! tab !                        the ! characters show where tabs are
#string of characters#
! tab !                        the ! characters show where tabs are
.fc
```

and here is the output after formatting:

```
!                                  !
string of characters
!                                  !
```

This is not very exciting — the characters in the field are simply left-adjusted in the field, and the rest of the field up to the tab stop are padded with spaces. You would get exactly the same result if you placed the padding indicator character at the right end of the field to indicate that you wanted the padding on the right:

```
.ta 3.0i                     set a single tab at three inches
.fc # @                      set field delimiter character to #
                             set padding indicator character to @
! tab !                           the ! characters show where tabs are
#string of characters@#
! tab !                           the ! characters show where tabs are
.fc
```

As you can see, the result is identical to the one just above:

```
!                                        !
string of characters
!                                        !
```

But now we can place a padding indicator character at the left end of the field and get strings right-adjusted in the field:

```
.ta 3.0i                     set a single tab at three inches
.fc # @                      set field delimiter character to #
                             set padding indicator character as @
! tab !                           the ! characters show where tabs are
#@string of characters#
#@another string of characters#
! tab !                           the ! characters show where tabs are
.fc
```

We used two strings of different length here to show how they are right-adjusted against the tab stop:

```
!                                        !
                        string of characters
                 another string of characters
!                                        !
```

You can see how the spaces were placed on the left end of the field because that is we where we placed the padding indicator character, and the strings got adjusted right to the tab stop.

Then we can get fields centered by placing the padding indicator character at both ends of the string:

```
.ta 3.0i                     set a single tab at three inches
.fc # @                      set field delimiter character to #
                             set padding indicator character as @
! tab !                           the ! characters show where tabs are
#@string of characters@#
#@longer string of characters@#
! tab !                           the ! characters show where tabs are
.fc
```

Again we used two strings of different lengths to show the effect of centering the field:

```
!                                              !
             string of characters
         longer string of characters
!                                              !
```

In general, a field or a sub-field *between* a pair of padding indicator characters is centered in its space on the line.

Things get even more useful when you have multiple sub-fields in a field — the padding spaces are then parceled out so that the sub-fields are uniformly left-adjusted, right-adjusted, or centered between the current position and the next tab stop:

```
.ta 5.0i                    set a single tab at five inches
.fc # @                     set field delimiter character to #
                            set padding indicator character as @
! tab !                          use the ! characters to show where tabs are
#string of characters#
#string of characters@another string#
! tab !                          use the ! characters to show where tabs are
```

and here is the output after we format that:

```
!                                                              !
string of characters
string of characters                             another string
!                                                              !
```

And finally we can show three strings within a field, with the left part left-adjusted, the center part centered, and the right part right-adjusted:

```
.ta 5.0i
.fc # @
! tab !
#left string@center string@right string#
#longer left string@longer center string@longer right string#
! tab !
```

and here is the output after we format that:

```
!                                                              !
left string              center string              right string
longer left string    longer center string    longer right string
!                                                              !
```

So to summarize, a field is contained between a pair of field delimiter characters. A field consists of sub-fields 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 sub-fields and the field length is incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding can be negative.

### *Summary of the* `.fc` *Request*

| | |
|---|---|
| *Mnemonic:* | field character |
| *Form of Request:* | `.fc` *f p* |
| *Initial Value:* | Field mechanism is off |
| *If No Argument:* | Field mechanism is turned off. |
| *Explanation:* | Set the field delimiter to *f*; set the padding indicator to *p* (if specified) or to the *space* character if *p* is not specified. In the absence of arguments, the field mechanism is turned off. |

7

# Titles and Page Numbering

# 7

# Titles and Page Numbering

## 7.1. Titles in Page Headers

This is an area where things get tougher, because `troff` doesn't do any of this automatically. Of necessity, some of this section is a cookbook, to be copied literally until you get some experience.

Suppose you want a title at the top of each page, saying just

left top    center top    right top

There was a very early text formatter called *roff*, where you could say

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

to get headers and footers automatically on every page. Alas, this doesn't work in `troff`, which is a serious hardship for the novice. Instead you have to do a lot of specification:

- You have to say what the actual title is (reasonably easy — you just use the `.tl` request to specify the title).
- You have to specify when to print the title (also reasonably easy — you set a trap to call a macro that actually does the work),
- and finally you have to say what to do at and around the title line (this is the hard part).

Taking these three things in reverse order, first we define a `.NP` macro (for new page) 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 make sure we're at the top of a page, we issue a 'begin page' request `'bp`, which skips to top-of-page (we'll explain the `'` shortly). Then we space down half an inch (with the `'sp 0.5i` request), and print the title (the use of `.tl` should be self explanatory — later we will discuss the title parameters), space another 0.3 inches (with the 'sp 0.3i request), and we're done.

To ask for .NP at the bottom of each page, we have to say something like 'when the text is within an inch of the bottom of the page, start the processing for a new page'. This is done with a 'when' request .wh:

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

See Chapter 10 for a more detailed description of the .wh request. No dot (.) is used before NP in the when request because in this case, we're specifying the name of a macro, not calling a macro. The minus sign means measure up from the bottom of the page, so '-1i' means one inch from the bottom.

The .wh request appears in the input outside the definition of .NP; typically the input would be

```
.de NP
definition of the NP macro
..
.wh -1i NP
```

Now what happens? As text is actually being output, troff keeps track of its vertical position on the page. After a line is printed within one inch from the bottom, the .NP macro is activated. In the jargon, the .wh request sets a trap at the specified place, which is 'sprung' when that point is passed. .NP skips to the top of the next page (that's what the 'bp was for), then prints the title with the appropriate margins.

Why 'bp and 'sp instead of .bp and .sp? The answer is that .bp and .sp, like several other requests, *break* the current line — that is, 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. If we had used .bp or .sp in the .NP macro, a break would occur in the middle of the current output line when a new page is started. The effect would be to print the left-over part of that line at the top of the page, followed by the next input line on a new output line, something like this:

```
last line but one at almost the bottom of the page
last line at the bottom of the

title on the bottom of the page

```

*page break*

*title on the top of the next page*

```
page.
```

This is not what we want. Using ′ instead of `.` for a request tells `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.

The list of requests that break lines is short and natural:

Table 7-1 *Requests that Cause a Line Break*

| *Command* | *Explanation* |
|---|---|
| `.bp` | Begin a new page |
| `.br` | Break the current output line |
| `.ce` | Center line(s) |
| `.fi` | Start filling text lines |
| `.nf` | Stop filling text lines |
| `.sp` | Space vertically |
| `.in` | Indent the left margin |
| `.ti` | Temporary indent the left margin for the next line only |

No other requests break lines, regardless of whether you use a `.` or a ′. If you really *do* need a break, add a `.br` (break) request at the appropriate place.

## 7.2. Fonts and Point Sizes in Titles

One other thing to beware of — if you're changing fonts or point sizes a lot, 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` (length of title) request.

***Summary of the*** `.lt` ***Request***

| | |
|---|---|
| *Mnemonic:* | length of title |
| *Form of Request:* | `.lt` ±*N* |
| *Initial Value:* | 6.5 inches |
| *If No Argument:* | Previous value |
| *Explanation:* | Set length of title to ±*N*. The line-length and the title-length are *independent*. Indents do not apply to titles; page-offsets do. |
| *Notes:* | E, m (see Table A-2) |

There are several ways to fix the problems of point sizes and fonts in titles. For the simplest applications, we can define the `.NP` macro to set the proper size and font for the title, then restore the previous values, like this:

```
.de NP
'bp
'sp 0.5i
.ft R     \" set title font to Roman
.ps 10    \" 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` request contain size or font changes. What we would like to do in cases like this is remember the status of certain aspects of the environment, change them to meet our needs for the time being, and then restore them after we're done with the special stuff. This requirement is satisfied by `troff`'s environment mechanism discussed in Chapter 17, *Environments*.

To get a footer at the bottom of a page, you can modify `.NP` so it does some processing before the `'bp` request, or split the job so that there is a separate footer macro invoked at the bottom margin and a header macro invoked at the top of the page.

Output page numbers are computed automatically as each page is produced (starting at 1), but no numbers are printed unless you ask for them explicitly. 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.

## 7.3. `.pc` — Page Number Character

You can change the page number character with the `.pc` request.

***Summary of the*** `.pc` ***Request***

| | |
|---|---|
| *Mnemonic:* | page-number character |
| *Form of Request:* | `.pc` *c* |
| *Initial Value:* | `%` |
| *If No Argument:* | Off |
| *Explanation:* | Set the page-number character to *c*, or remove it if there is no *c* argument. The page-number register remains `%`. |

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 skip to the new page. Again, `.bp` +*n* sets the page number to *n* more than its current value; `.bp` means `.bp +1`.

## 7.4. `.tl` Request — Three Parameters

The `.tl` (title) request automatically places three text fields at the left, center, and right of a line (with a title-length specifiable via the `.lt` (length of title) request. The most common use for three-part titles is to put running headers and footers at the top and bottom of pages just like those in this manual. In fact, the `.tl` request may be used anywhere, and is independent of the normal text collecting process. For example, we just placed a three-part title right here in the text:

Hunting the Snark – 85 – Smiles and Soap

by typing the a three-part title request that looks like:

```
.tl 'Hunting the Snark'- % -'Smiles and Soap'
```

and you might notice that the page number in the formatted example is the same as the page number for this page.

***Summary of the*** `.tl` ***Request***

| | |
|---|---|
| *Mnemonic:* | title |
| *Form of Request:* | `.tl` *'left'center'right'* |
| *Initial Value:* | Nothing |
| *If No Argument:* | Nothing |
| *Explanation:* | The strings in the *left*, *center*, and *right* fields are respectively left-adjusted, centered, and right-adjusted in the current title-length. Any of the strings may be empty, and overlapping is permitted. If the page-number character (initially `%`) is found within any of the fields it is replaced by the current page number having the format assigned to register `%`. Any character may be used as the string delimiter. |

# 8

# troff Input and Output

# 8

# troff Input and Output

We now describe two troff requests that we omitted earlier, because their usefulness is more apparent when you understand the troff command line. Normally troff takes its input from the files given when it is called up. However there are ways in which the formatter can be made to take part of its input from elsewhere, using troff requests embedded in the document text.

## 8.1. .so — Read Text from a File

The .so request, which tells troff to switch over and take its source from the named file. For example, suppose you have a set of macros that you have defined, and you have them in a file called *macros*. We can call them up from the troff command line:

```
hostname% troff macros document
hostname%
```

as we showed earlier, but it's a bit of a nuisance having to do this all the time. Also, if only some of our documents use the macros, and others don't, it can be difficult to remember which is which. An alternative is to make the first line of the *document* file look like this:

```
.so macros
```

Now we can format the document by:

```
hostname% troff document
hostname%
```

The first thing troff sees in the file *document* is the request .so macros which tells it to read input from the file called *macros*. When it finishes taking input from *macros*, troff continues to read the original file *document*.

Another way of using the .so request lets you format a complete document, held in several files, by only giving one filename to the troff command. Let us create a file called *document* containing:

```
.so macros
.so section.1
.so section.2
.so section.3
          and so on through the document until . . .
.so appendix.C
```

We can now format it with the troff command line:

```
hostname% troff document | lpr
hostname%
```

This is a lot easier than typing all the filenames each time you format the document, and a lot less prone to error.

This technique is especially useful if your filenames reflect the contents of the various sections, rather than the order in which they appear. For instance, look at this file which describes a whole book (something like the one you are reading):

```
hostname% cat book
.so bookmacros
.so preface
.so intro
.so login      \"Getting Started on the UNIX System
.so directs    \"Directories and the File System
.so stdio      \"Commands, Processes, and Standard Files
          <etc...>
.so biblio     \"Bibliography
hostname%
```

It is obviously much easier to format the whole thing with a troff command line like this:

```
hostname% troff book | lpr
hostname%
```

than it would be if you had to supply all the filenames in the right order. Notice that we used the comment feature of troff to tie chapter titles to filenames.

***Summary of the* .so *Request***

| | |
|---|---|
| *Mnemonic:* | source |
| *Form of Request:* | .so *filename* |
| *Explanation:* | Switch source file — the top input (file reading) level is switched to *filename*. The sourced-in file is read directly and processed immediately when the .so line is encountered. When the new file ends, input is again taken from the original file. .sos may be nested. |

### 8.2. .nx — Read Next Source File

***Summary of the* .nx *Request***

| | |
|---|---|
| *Mnemonic:* | next |
| *Form of Request:* | .nx *filename* |
| *If No Argument:* | end-of-file |
| *Explanation:* | Next file is *filename*. The current file is considered ended, and the input is immediately switched to *filename*. There is no return to the file containing the .nx command. |

### 8.3. Pipe Output to a Specified Program (nroff only)

A couple of examples of programs you might want you pipe your nroff output to are lpr and col. Your source line might look like this:

```
.pi /usr/ucb/lpr
```

or

```
.pi /usr/bin/col
```

if you had formatted tables in your source file.

***Summary of the* .pi *Request***

| | |
|---|---|
| *Mnemonic:* | pipe |
| *Form of Request:* | .pi *program_name* |
| *Explanation:* | Pipe output to *program* (nroff only). This request must occur *before* any printing occurs. No arguments are transmitted to *program.* |

## 8.4. .rd — Read from the Standard Input

Another troff request that switches input from the file you specify is the .rd (read) request. The *standard input* can be the user's keyboard, a pipe, or a file. The .rd request reads an insertion from the standard input. When troff encounters the .rd request, it prompts for input by sounding the terminal bell or flashing the screen. A visible prompt can be given by adding an argument to .rd, as we show in the example below.

Everything typed up to a blank line (two newline characters in a row) is inserted into the text being formatted at that point. This can be used to 'personalize' form letters. If you have an input file with this text:

```
.po 10
.nf
.in 20
14th February
.in 0
Dear
.rd who
     Will you be my Valentine?
     If you will, give me a sign
     (I like roses, I like wine).
```

then when you format it, you will be prompted for input:

```
hostname% troff valentine | lpr
who:Peter

hostname%
```

After typing the name Peter you have to press the RETURN key twice, since troff needs a blank line to end input. The result of formatting that file is:

```
                              14th February


          Dear
          Peter
               Will you be my Valentine?
               If you will, give me a sign
               (I like roses, I like wine).
```

To get another copy of this for Bill, you just run the troff command again:

```
hostname% troff valentine | lpr
who:Bill

hostname%
```

and again for Joe, and for Manuel, and Louis, and Alphonse, and ...

Since troff takes input from the terminal up to a blank line, you are not limited to a single word, or even a single line of input. You can use this method to insert addresses or anything else into form letters.

### *Summary of the* .rd *Request*

| | |
|---|---|
| *Mnemonic:* | read |
| *Form of Request:* | .rd *prompt* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | *prompt*=BEL |
| *Explanation:* | Read insertion from the standard input until two newlines in a row are found. If the standard input is the user's keyboard, *prompt* (or a BEL) is written onto the user's terminal. .rd behaves like a macro, and arguments may be placed after *prompt*. Use the standard way to access arguments in macros (see Chapter 10. |

If insertions are to be taken from the terminal keyboard while output is being printed on the terminal, the command line option –q will turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input cannot simultaneously come from the standard input.

As an example, multiple copies of a form letter may be prepared by entering the insertions for all the copies in one file to be used as the standard input, and causing the file containing the letter to reinvoke itself using .nx (see the previous section); the process would ultimately be ended by a .ex in the insertion file. Example:

```
Letter File               Names File
Dear ...                      John
.rd                       blank line
  .                       Bill
  .                       blank line
  .                       .ex
.nx Letter
```

To put everything together, you could use:

```
hostname% cat Names | troff Letter
```

### 8.5. .ex — Exit from nroff or troff

***Summary of the* .ex *Request***

| | |
|---|---|
| *Mnemonic:* | exit |
| *Form of Request:* | .ex *prompt* |
| *Explanation:* | Exit from nroff or troff. Text processing is terminated exactly as if all input had ended. |

### 8.6. .tm — Send Messages to the Standard Error File

The .tm (terminal message) request displays a message on the standard error file. The request looks like:

```
.tm tell me some good news
```

and when troff or nroff encounters this in the input file, it displays the string

```
tell me some good news
```

on the standard error file. This request has been used in older versions of the –ms macro package to rebuke the user when (for instance) an abstract for a paper was longer than a page. Other macro packages use the .tm request for assisting in generating tables of contents and indices and such supplementary material.

***Summary of the* .tm *Request***

| | |
|---|---|
| *Mnemonic:* | terminal message |
| *Form of Request:* | .tm *string* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Display a newline |
| *Explanation:* | After skipping initial blanks, *string* (rest of the line) is read in *copy mode* and written on the user's terminal. |

# 9

# Strings

# 9

# Strings

Obviously if a paper contains a large number of occurrences of an acute accent over a letter 'e', typing `\o"e\'"` for each é would be a great nuisance. (See Chapter 12 for more detailed information on drawing lines and characters.

Fortunately, `troff` provides a way that you can store an arbitrary collection of text in a string, and thereafter use the string name as a shorthand for its contents. Strings are one of several `troff` mechanisms whose judicious use lets you type a document with less effort and organize it so that extensive format changes can be made with few editing changes. A reference to a string is replaced in the text by the string definition.

A string is a named sequence of characters, *not* including a newline character, that may be interpolated by name at any point in your text. Note that names of `troff` requests, names of macros, and names of strings all share the same name list. String names may be one or two characters long and may usurp previously-defined request, macro, or string names.

You create a string (and give it an initial value) with the `.ds` (define string) request. You can later add more characters to the end of the string by using the `.as` (append to string) request.

String names may be either one or two characters long. You get the value of a string placed in the text, where it is said to be interpolated, by using the notation:

```
\*x
```

for a one-character string named *x*, and the more complicated notation:

```
\*(xx
```

for a two-character string named *xx*.

String references and macro invocations may be nested.

## 9.1. .ds — Define Strings

You create a string (and define its initial value) with the .ds (define string) request. The line

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

defines the string e to have the value \o"e\´"

You refer to them with the sequence * *x* for one-character names or * ( *xy* for two-character names. Thus, to get téléphone, given the definition of the string e as above, we can say t*el*ephone.

As another live example, in the section on ligatures in Chapter 5, *Fonts and Special Characters*, we noted that troff doesn't know about the Scandinavian ligatures — you have to decide for yourself how to define them. Here are our definitions of the strings for those ligatures:

```
.ds ae a\h'-(\w'a'u*4/10)'e
.ds Ae A\h'-(\w'A'u*4/10)'E
.ds oe o\h'-(\w'o'u*4/10)'e
.ds Oe O\h'-(\w'O'u*4/10)'E
```

See the section entitled "\h Function — Arbitrary Horizontal Motion" in Chapter 12 for a discussion on what the \h constructs are doing in the string definitions above. Having defined the strings, all you have to do is type the string references like this:

```
. . . the Scandinavian ligatures \*(oe, \*(ae, \*(Oe, and \*(Ae . . .
```

in order to get . . . the Scandinavian ligatures œ, æ, Œ, and Æ . . . into your stream of text.

If a string must begin with spaces, define it as

```
.ds xx "      text
```

The double quote character signals the beginning of the definition. 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, the backslash and the newline characters are disregarded resulting in the next line being added to the current one. So you can make a long string simply by ending each line except the last with a backslash:

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

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

***Summary of the* .ds *Request***

| | |
|---|---|
| *Mnemonic:* | define string |
| *Form of Request:* | .ds *xx string* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignored |
| *Explanation:* | Define a string *xx* containing *string*. Any initial double-quote in *string* is stripped off to permit initial spaces. |

## 9.2. .as — Append to a String

The .as (append to string) request adds characters to the end of a string. You use the .as request like this:

```
.as  xx string-of-characters
```

where *string-of-characters* is appended to the end of whatever is already in the string *xx*.

Note that the string mentioned in a .as request is created if it didn't already exist, so in that respect an initial .as request acts just like a .ds request.

For example, here's a short fragment from the .H macro that was used to generate the section numbers in this document. The .H macro is called up like

```
.H level-number  "Text of the Title"
```

where level-number is 1, 2, 3, ... to indicate that this is a first, second, third, ... level heading. The .H macro keeps track of the various section numbers via a bunch of number registers H1 through H5, and they are tested for and appended to the SN string if appropriate. For example:

```
.ds SN \\n(H1.                     set the initial section number string
.if \\n(NS>1 .as SN \\n(H2.        append H2 if needed
.if \\n(NS>2 .as SN \\n(H3.        append H3 if needed
.if \\n(NS>3 .as SN \\n(H4.        append H4 if needed
.if \\n(NS>4 .as SN \\n(H5.        append H5 if needed
        .
        .
   more processing to compute indentations and such ...
        .
        .
\\*(SN\\ \\ \t\c                   Now output the text
\&\\$2
        .
        .
   and yet more processing ...
        .
        .
```

Let's unscramble that mess. The essential parts are the initial line that says:

```
.ds SN \\n(H1.                     set the initial section number string
```

which sets the SN (section number) string to the value of the H1 number register that counts chapter level numbers. Then the following four lines essentially all perform a test that says:

> .if the *level-number* is greater than *N*, append the next higher section counter to the string. That is, if the current section number is greater than 2, we append the value of the level 3 counter, then if the section number is greater than 3, we append the value of the level 4 counter, and so on.

Finally, the built-up SN string, followed by the text of the title, gets placed into the output text with the lines that read:

```
\\*(SN\\ \\ \t\c                   Now output the text
\&\\$2
```

And in fact we can use the mechanisms that exist to play games like that because we are using a macro package to format this document, and those number registers are available to us. So we can define a string like this:

```
.ds XX \n(H1.
```

and interpolate that string like this:

```
\*(XX
```

to get the value

9.

printed in the text. Now we can append the rest of the section counters to that *XX* string like this (without caring whether they have any values):

```
.as XX \n(H2.\n(H3.\n(H4.\n(H5.
```

and then when we interpolate that string we get this:

9.2.0.0.0.

which, if you look, should be the section number of the stuff you are now reading.

***Summary of the* `.as` *Request***

| | |
|---|---|
| *Mnemonic:* | append to string |
| *Form of Request:* | `.as` *xx string* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignored |
| *Explanation:* | Append *string* to string *xx* (append version of `.ds`). The string *xx* is created if it didn't already exist. |

## 9.3. Removing or Renaming String Definitions

Strings (just like macros) can be renamed with the `.rn` (rename) request, or can be removed from the namelist with the `.rm` (remove) request. Refer to Chapter 10 for more detailed descriptions of the `.rn` and `.rm` commands.

# 10

# Macros, Diversions, and Traps

# 10

# Macros, Diversions, and Traps

## 10.1. Macros

Before we can go much further in nroff or troff, we need to learn something about the macro facility. In its simplest form, a macro is just shorthand notation similar to a string. A macro is a collection of several separate troff commands which, when bundled together, achieves (sometimes complex) formatting when the macro is invoked. Whereas a string is somewhat limited because its definition is specific, a macro can interpret arguments that can change its behavior from one invocation to the next.

A macro is a named set of arbitrary lines that may be invoked by name or with a trap. Macros are created by .de and .di requests, and appended to by .am and .da requests; .di and .da requests cause normal output to be stored in a macro. A macro is invoked in the same way as a request; a control line beginning .*xx* interpolates the contents of macro *xx*. The remainder of the line may contain up to nine ***arguments***. Request, macro, and string names share the *same* name list. Macro names may be one or two characters long and may usurp previously-defined request, macro, or string names. String references and macro invocations may be nested. Any of these entities may be renamed with a .rn request or removed with a .rm request.

### .de — Define a Macro

Suppose we want every paragraph to start in exactly the same way — with a space and a temporary indent of two ems. We show a (very simplified) version of the .PP (paragraph) macro from the **–ms** macro package:

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

Then to save typing, we would like to collapse these into one shorthand line, a troff 'request' like

```
.PP
```

that would be treated by troff exactly as if you had typed:

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

.PP is called a macro. The way we tell troff what .PP means is to define it with the .de (define) request:

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

The first line names the macro (we used .PP) which is a standard macro notation for 'paragraph'. It is common practice to use upper-case names for macros so that their names don't conflict with ordinary troff requests. The last line .. marks the end of the definition. In between the beginning and end of the definition, is the text (often called the *replacement text*), which is simply inserted whenever troff sees the request or macro call

```
.PP
```

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

Using macros for commonly-occurring sequences of requests is critically important. Not only does it save typing, but it makes later changes much easier. Suppose we decide that the paragraph indent should be greater, the vertical space should be less, and the font should be Roman. Instead of changing the whole document, we need only change the definition of the .PP macro to something like

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

and the change takes effect everywhere we used .PP.

The notation \" is an in-line troff function that means that the rest of the line is to be ignored. We use it here to add comments to the macro definition (a wise idea once definitions get complicated).

***Summary of the*** `.de` ***Request***

| | |
|---|---|
| *Mnemonic:* | define |
| *Form of Request:* | `.de` *xx yy* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | .*yy*=`. .` |
| *Explanation:* | 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 '`. .`'. A macro may contain `.de` requests provided the terminating macros differ or the contained definition terminator is concealed. '`. .`' can be concealed as `\\. .` which will copy as `\. .` and be reread as '`. .`'. |

### `.rm` — Remove Requests, Macros, or Strings

***Summary of the*** `.rm` ***Request***

| | |
|---|---|
| *Mnemonic:* | remove |
| *Form of Request:* | `.rm` *xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignored |
| *Explanation:* | 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 — Rename Requests, Macros or Strings

| | ***Summary of the* `.rn` *Request*** |
|---|---|
| *Mnemonic:* | rename |
| *Form of Request:* | `.rn` *xx yy* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignored |
| *Explanation:* | Rename request, macro, or string *xx* to *yy*. If *yy* exists, it is removed first. |

Refer to Chapter 9, *Strings* for information on defining strings.

As another example of macros, consider these two, which start and end a block of offset, unfilled text, like most of the examples in this paper:

```
.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 like

Copy to:
John Doe
Richard Roberts
Stanley Smith

by the requests `.BS` and `.BE`, and it will come out as it did above. Notice that we indented by an incremental amount: `.in +0.3i` instead of `.in 0.3i`. This way we can nest our uses of `.BS` and `.BE` to get blocks within blocks.

If later on we decide that the indent should be half an inch, then it is only necessary to change the definitions of `.BS` and `.BE`, not the whole paper.

### Macros With Arguments

The next step is to define macros that can change from one use to the next according to parameters supplied as arguments to the macro. To make this work, we need two things: first, when we define the macro, we have to indicate that some parts of it will be provided as arguments when the macro is called. Then when the macro is called we have to provide actual arguments to be plugged into the definition.

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

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

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

and called by

```
.xx Monday 14th
```

to produce the text

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

Note that the \$ was concealed in the definition with a preceding backslash (\). The number of currently available arguments is in the .$ register.

No arguments are available at the top (non-macro) 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.

Let's illustrate by defining a macro .SM that will print its argument two point sizes smaller than the surrounding text. That is, the macro call

```
.SM UNIX
```

will produce UNIX.

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.

As a slightly more complicated version, 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, so

```
.SM  UNIX  ),
```

produces

UNIX),

while

```
.SM  UNIX  ).  (
```

produces

(UNIX).

It is convenient to reverse the order of arguments because trailing punctuation is much more common than leading.

The following macro `.BD` is the one used to make the 'bold Roman' we have been using for `troff` request 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 discuss in the section *Copy Mode Input Interpretation.* The `\&` is there in case the argument begins with a period.

Two backslashes are needed with the `\\$`*n* commands, though, to protect one of them when the macro is being defined. Perhaps a second example will make this clearer. Consider a macro called `.SH` which produces section headings like the ones in this manual, with the sections numbered automatically, and the title in bold in a smaller size. The use is

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

If the argument to a macro is to contain spaces, then it must be surrounded by double quotes, unlike a string, where only the leading quote is permitted.

Here is the definition of the .SH macro:

```
.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. 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 called. If that's what you want, fine, but that isn't the case here. 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 our .NP macro which had:

```
.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 for left title, center title, and right title, respectively. If these are empty, then the title will be a blank line. Normally CT would be set with something like

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

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

### .am — Append to a Macro

| | *Summary of the* .am *Request* |
|---|---|
| *Mnemonic:* | append to macro |
| *Form of Request:* | .am *xx yy* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | .*yy*=. . |
| *Explanation:* | Append to macro *xx* (append version of .de). |

### Copy Mode Input Interpretation

During 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 preceded by backslash (\ newline) are eliminated.
- Comments indicated by \" are eliminated.
- \t and \a are interpreted as ASCII horizontal tab and SOH respectively (see Chapter 6, *Tabs, Leaders, and Fields* for more information).
- \\ is interpreted as \
- \. is interpreted as "."

These interpretations can be suppressed by adding another \ (backslash) to the beginning of the command. For example, since \\ maps into a \, \\n will copy as \n which will be interpreted as a number register indicator when the macro or string is reread.

## 10.2. Using Diversions to Store Text for Later Processing

There are numerous occasions in page layout when it is necessary to store some text for a period of time without actually printing it. Footnotes are the most obvious example: the text of the footnote usually appears in the input well 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, which implies that there must be a way to process the footnote at least enough to decide its size without printing it.

troff provides a mechanism called a diversion for doing this processing. A diversion is very similar to a macro and in fact uses the same mechanisms as the macro facility. Any part of the output may be sent into a diversion instead of being printed, and then at some convenient time the diversion may be brought back into the input.

## .di — Divert Text

The request .di *xy* begins a diversion — all subsequent output is collected into the diversion called *xy* until a .di request with no argument is encountered, which terminates the diversion. The processed text is available at any time thereafter, simply by giving the request:

```
.xy
```

The vertical size of the last finished diversion is contained in the built-in number register dn.

As a simple example, suppose we want to implement a 'keep-release' operation, so that text between the requests .KS and .KE will not be split across a page boundary (as for a figure or table). 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. So:

```
.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. dn is the amount of text in the diversion; .t (another built-in register) 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. In either case, the diverted output is then brought back with It.XX. troff will do no further processing on it.

This is not the most general keep-release, nor is it robust in the face of all conceivable inputs, but it would require more space than we have here to write it in full generality. This section is not intended to teach everything about diversions, but to sketch out enough that you can read existing macro packages with some comprehension.

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 nofill mode regardless of the current *V*. Constant-spaced (.cs) or emboldened (.bd) text that is diverted can be reread correctly only if these modes are again or still in effect at reread time. One way to do this is to embed in the diversion the appropriate .cs or .bd requests with the 'transparent' mechanism described in the chapter *Introduction to nroff and troff*.

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

| | ***Summary of the* .di *Request*** |
|---|---|
| *Mnemonic:* | divert |
| *Form of Request:* | .di *xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | End of diversion |
| *Explanation:* | 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. |
| *Notes:* | D (see Table A-2) |

### .da — Append to a Diversion

| | ***Summary of the* .da *Request*** |
|---|---|
| *Mnemonic:* | append to diversion |
| *Form of Request:* | .da *xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | End of diversion |
| *Explanation:* | Append to diversion *xx*. This is the diversion equivalent of the .am (append to macro) request. |

## 10.3. Using Traps to Process Text at Specific Places on a Page

Three types of trap mechanisms are available, namely *page traps*, *diversion traps*, and *input-line-count traps*.

Macro-invocation traps may be planted using the .wh (when) request at any page position including the top. This trap position may be changed using the .ch (change) request. Trap positions at or below the bottom of the page have no effect unless or until moved to within the page or rendered effective by an increase in page length.

Two traps may be planted at the *same* position only by first planting them at different positions and then moving one of the traps; the first planted trap will conceal the second unless and until the first one is moved. If the first 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 the `.dt` (diversion trap) request. 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 the `.it` request below.

### `.wh` — Set Page or Position Traps

| | ***Summary of the*** `.wh` ***Request*** |
|---|---|
| *Mnemonic:* | when |
| *Form of Request:* | `.wh` *N xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Not applicable |
| *Explanation:* | Install a trap to invoke *xx* at page position *N;* a *negative N* is interpreted with respect to the page bottom. Any macro previously planted at *N* is replaced by *xx*. A zero *N* refers to the top of a page. In the absence of *xx*, the first-found trap at *N*, if any, is removed. |
| *Notes:* | v (see Table A-2) |

### `.ch` — Change Position of a Page Trap

| | ***Summary of the*** `.ch` ***Request*** |
|---|---|
| *Mnemonic:* | change trap |
| *Form of Request:* | `.ch` *xx N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Not applicable |
| *Explanation:* | Change the trap position for macro *xx* to be *N*. In the absence of *N*, the trap, if any, is removed. |
| *Notes:* | v (see Table A-2) |

### `.dt` — Set a Diversion Trap

| | ***Summary of the*** `.dt` ***Request*** |
|---|---|
| *Mnemonic:* | diversion trap |
| *Form of Request:* | `.dt` *N xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Turn off diversion trap |
| *Explanation:* | 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. |
| *Notes:* | D, v (see Table A-2) |

### `.it` — Set an Input-Line Count Trap

| | ***Summary of the*** `.it` ***Request*** |
|---|---|
| *Mnemonic:* | input-line-count trap |
| *Form of Request:* | `.it` *N xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Turn off trap |
| *Explanation:* | 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. |
| *Notes:* | E (see Table A-2) |

### `.em` — Set the End of Processing Trap

| *Summary of the* `.em` ***Request*** | |
|---|---|
| *Mnemonic:* | end macro |
| *Form of Request:* | `.em` *xx* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | No trap installed |
| *Explanation:* | Call the macro *xx* 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. |

# 11

# Number Registers

# 11

# Number Registers

In a programmable text formatter such as `troff`, you need a facility for storing numbers somewhere, retrieving the numbers, and for doing arithmetic on those numbers. `troff` meets this need by providing things called *number registers*. Number registers give you the ability to define variables where you can place numbers, retrieve the values of the variables, and do arithmetic on those values. Number registers, like strings and macros, can be useful in setting up a document so it is easy to change later. And of course number registers serve for any sort of arithmetic computation.

Number registers, just like strings, have one- or two-character names. They are set by the `.nr` (number register) request, and are referenced anywhere by `\n` *x* (one-character name) or `\n(` *xy* (two-character name). When you access a number register so that its value appears in the printed text, the jargon says that you have *interpolated* the value of the number register.

A variety of parameters are available to the user as predefined, named number registers (see Appendix D). In addition, users may define their 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.

`troff` defines several pre-defined number registers listed in Appendix D. Among them are `%` for the current page number, `nl` for the current vertical position on the page, `dy`, `mo`, and `yr` for the current day, month and year (see Table D-1) for a complete list); and `.s` and `.f` for the current size and font — the font is a number from 1 to 4. Any of these number registers can be used in computations like any other register, but some, like `.s` and `.f`, cannot be changed with a `.nr` request because they are "read only" (see Table D-2) for a complete list).

## 11.1. `.nr` — Set Number Registers

You create and modify number registers using the `.nr` (number register) request. In its simplest form, the `.nr` request places a value into a number register — the register is created if it doesn't already exist. The `.nr` request specifies the name of the number register, and also specifies the initial value to be placed in there. So the request

```
.nr PD 1.5v
```

would be a request to set a register called PD (which we might know as 'Paragraph Depth' if we were writing a macro package) to the value 1.5v (1.5 of troff's vertical units).

As an example of the use of number registers, in the **–ms** macro package, most significant parameters are defined in terms of the values of a handful of number registers (see the chapter "Formatting Documents with the -ms Macros" in *Formatting Documents*). 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, a user may say:

```
.nr PS 10
.nr VS 12
```

The paragraph macro .PP is defined (roughly) as follows:

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

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

Why are there two backslashes? When troff originally reads the macro definition, it peels off one backslash to see what's coming next. To ensure that another is left in the definition when the macro is *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 the macro is used.

Protecting by an extra layer of backslashes is only needed for \n, *, \$, and \ itself. Things like \s, \f, \h, \v, and so on do not need an extra backslash, since they are converted by troff to an internal code immediately upon being seen.

| ***Summary of the*** `.nr` ***Request*** | |
|---|---|
| *Mnemonic:* | number register |
| *Form of Request:* | `.nr` *R* ±*N M* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignored |
| *Explanation:* | Assign the value ±*N* to number register *R*, with respect to the previous value, if any. Set the increment for auto-incrementing to *M*. |
| *Notes:* | u (see Table A-2) |

## 11.2. Auto-Increment Number Registers

When you set a number register with the `.nr` request, you can also specify an additional number as an auto-increment value — that is, the number is added to the number register every time you access the number register. You specify the auto-increment value with a request such as:

```
.nr sn 0 1
```

to specify a (hypothetical) section number register that starts off with the value 0 and is incremented by 1 every time you use it. This might be applicable (for instance) to numbering the sections of a document automatically — something you might expect a computer to do for you. You might also define a numbered list macro that would clock up the item number every time you added a new list item.

Here's a very quick and dirty example of the use of auto-incrementing a number register:

```
.nr cn -1 2
. . .
the odd numbers \n+(cn, \n+(cn, \n+(cn, \n+(cn, \n+(cn, \n+(cn,
. . .
```

When we format the above sequence, we get the following:

> ... the odd numbers 1, 3, 5, 7, 9, 11, ...

The table below shows the effects of accessing the number registers *x* and *xx* after a `.nr` request that sets them to the value *N* with an auto-increment value of *M*.

Table 11-1 *Access Sequences for Auto-incrementing Number Registers*

| *Request* | *Access Sequence* | *Effect on Register* | *Value Interpolated* |
|---|---|---|---|
| .nr *x* N M<br>.nr *xx* N M | \n*x*<br>\n(*xx* | none<br>none | *N*<br>*N* |
| .nr *x* N M<br>.nr *x* N M | \n+*x*<br>\n–*x* | *x* incremented by *M*<br>*x* decremented by *M* | *N+M*<br>*N–M* |
| .nr *xx* N M<br>.nr *xx* N M | \n+(*xx*<br>\n–(*xx* | *xx* incremented by *M*<br>*xx* decremented by *M* | *N+M*<br>*N–M* |

## 11.3. Arithmetic Expressions with Number Registers

Arithmetic expressions can appear anywhere that a number is expected. As a trivial example,

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

decrements the value in the PS macro by 2.

Expressions can use the arithmetic operators and logical operators as shown in the table below. Parts of an expression can be surrounded by parentheses.

Table 11-2 *Arithmetic Operators and Logical Operators for Expressions*

| *Arithmetic Operator* | *Meaning* |
|---|---|
| + | Addition |
| – | Subtraction |
| / | Division |
| * | Multiplication |
| % | Modulo |
| *Logical Operator* | *Meaning* |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| = or = = | Equal to |
| & | and |
| : | or |

Except where controlled by parentheses, evaluation of expressions is left-to-right — there is no operator precedence.

Although the arithmetic we have done so far has been straightforward, more complicated things are somewhat tricky. First, number registers hold only integers. `troff` arithmetic uses truncating integer division. Second, in the absence of parentheses, evaluation is done from 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 (but no spaces). Although integer division causes truncation, each number and its scale indicator is converted to machine units (1/432 inch) *before* any arithmetic is done, so 1i/2u evaluates to 0.5i correctly.

The scale indicator `u` often has to appear where you would not expect it — in particular, when arithmetic is being done in a context that implies horizontal or vertical dimensions. For example,

```
.ll 7/2i
```

would seem obvious enough — 3.5 inches. Sorry — remember that the default units for horizontal parameters like the `.ll` request are ems. So that expression is really '7 ems / 2 inches', and when translated into machine units, it becomes zero. How about

```
.ll 7i/2
```

Still no good — the '2' is '2 ems', so '7i/2' is small, although not zero. You must use

```
.ll 7i/2u
```

So again, a safe rule is to attach a scale indicator to every number, even constants.

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

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

does just what you want, so long as you don't forget the `u` on the `.ll` request.

## 11.4. `.af` — Specify Format of Number Registers

When you use a number register as part of the text, the contents of the register are said to be interpolated into the text at that point. For example, you could use the following sequence:

```
.nr xy 567
. . .
the value of the \fIxy\fP number register is: \n(xy.
. . .
```

and when you formatted that sequence, it would appear as:

> ... the value of the *xy* number register is: 567. ...

When interpolated, the value of the number register is read out as a decimal number. You can change this format by using the `.af` (assign format) request to get things like Roman numerals or sequences of letters. Here is the example of the auto-incrementing section above, but with the interpolation format now set for lower-case Roman numerals:

```
.nr cn -1 2
.af cn i
. . .
the odd Roman numerals \n+(cn, \n+(cn, \n+(cn, \n+(cn, \n+(cn, \n+(cn,
. . .
```

When we format the above sequence, we get the following:

> ... the odd Roman numerals i, iii, v, vii, ix, xi, ...

A decimal format having *N* digits specifies a field width of *N* digits.

Read-only number registers and the width function are always decimal.

The table below shows the different formats you can apply to a number register when it is interpolated.

Table 11-3 *Interpolation Formats for Number Registers*

| *Format* | *Description* | *Numbering Sequence* |
|---|---|---|
| `1` | decimal | 0, 1, 2, 3, 4, 5, ... |
| `001` | decimal with leading zeros | 000, 001, 002, 003, 004, 005, ... |
| `i` | lower-case Roman numerals | 0, i, ii, iii, iv, v, ... |
| `I` | upper-case Roman numerals | 0, I, II, III, IV, V, ... |
| `a` | lower-case letters | 0, a, b, c, ... aa, ab, ... aaa, ... |
| `A` | upper-case letters | 0, A, B, C, ... AA, AB, ... AAA, ... |

| | ***Summary of the*** `.af` ***Request*** |
|---|---|
| *Mnemonic:* | assign format |
| *Form of Request:* | `.af` *R c* |
| *Initial Value:* | Arabic |
| *If No Argument:* | Ignored |
| *Explanation:* | Assign format *c* to register *R*. |

### 11.5. `.rr` — Remove Number Registers

If you create many number registers dynamically, you may have to remove number registers that you aren't using any more to recapture internal storage space for newer registers. You remove a number register with the `.rr` (remove register) request:

```
.rr xy
```

removes the *xy* number register from the list.

| | ***Summary of the*** `.rr` ***Request*** |
|---|---|
| *Mnemonic:* | remove register |
| *Form of Request:* | `.rr` *R* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignored |
| *Explanation:* | 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. |

# 12

# Drawing Lines and Characters

# 12

# Drawing Lines and Characters

This section is a grab-bag of functions for moving to arbitrary places on the page and for drawing things. This section covers a number of useful topics:

- Local motions — how to move forward and backward and up and down on the page to get special effects.
- Constructing whole characters out of pieces of characters that are available in the special font — these facilities are for doing mathematical typesetting.
- Drawing horizontal and vertical lines to make boxes and underlines and such.
- Various types of padding characters, zero-width characters, and functions for obtaining the width of a character string.

Most of these commands are straightforward, but messy to read and tough to type correctly.

## 12.1. `\u` and `\d` Functions — Half-Line Vertical Movements

If you can't or don't want to use `eqn`, subscripts and superscripts are then most easily done with the half-line local motions `\u` (for up) and `\d` (for down). To move up the page half a point, insert a `\u` at the desired place, and to go down the page half a point, insert a `\d` at the desired place. The `\u` and `\d` in-line functions should always be used in pairs, as explained below. Thus if your input consists of the following fragment:

```
. . . area of a circle is 'Area = \(*pr\u2\d' when calculating . . .
```

the output when that fragment is formatted consists of:

... area of a circle is ‘Area = $\pi r^{2}$’ when calculating ...

This is a first approximation of what you want, but the superscript ‘2’ is too large. To make the ‘2’ smaller, bracket it with `\s−2...\s0`. This reduces the point-size by two points before the superscript and restores the point-size to the previous value after the superscript. This example input:

```
. . . area of a circle is 'Area = \(*pr\u\s−22\s0\d' when calculating . .
```

when formatted, generates:

. . . area of a circle is ‘Area = $\pi r^2$’ when calculating . . .

Now the reason that the \u and \d functions should always be correctly paired is that they refer to the *current* vertical spacing, so you must be sure to put any local motions either both inside or both outside any size changes, or you will get an unbalanced vertical motion. Carrying this example further, the input could look like this:

```
. . .  area of a circle is 'Area = \(*pr\u\s-22\d\s0' when calculating . . .
```

We'll format that example in a larger point-size so that you can see the effect of the baseline being out of whack. So when we format the above construct with the motions incorrectly paired, we get this:

. . . area of a circle is ‘Area = $\pi r^{2}$’ when calculating . . .

As you can see, the baseline is higher after the incorrectly-displayed equation.

## 12.2. Arbitrary Local Horizontal and Vertical Motions

The next two sections describe the in-line \v (vertical) and the \h (horizontal) local motion functions. The general form of these functions is \v'*N* ' for the vertical motion function, and \h'*N* ' for the horizontal motion function. The argument *N* in the functions is the distance to move. The distance *N* may be negative — the *positive* directions are *to the right* and *down*.

A *local* motion is one contained *within* a line. To avoid unexpected vertical dislocations, it is necessary that the *net* vertical local motion within a word in filled text, and otherwise within a line, be zero.

### \v Function — Arbitrary Vertical Motion

Sometimes the space given by \u and \d is not the right amount (usually too much). The in-line \v function requests an arbitrary amount of vertical motion. The in-line \v function

```
\v ' amount
```

moves up or down the page by the amount specified in *amount*. For example, here's how to get a large letter at the start of a verse:

```
.in +.3i
.ti -.3i
\v'1.0'\s36A\s0\v'-1.0'\h'-4p'wake! for Morning in the Bowl of Night
\h'2p'Has flung the Stone that puts the Stars to Flight:
.in -.3i
And Lo! the Hunter of the East has caught
The Sultan's Turret in a Noose of Light.
```

and when we format that verse we get:

Awake! for Morning in the Bowl of Night
Has flung the Stone that puts the Stars to Flight:
And Lo! the Hunter of the East has caught
The Sultan's Turret in a Noose of Light.[3]

The indent amount we used here (0.3 inch) was determined by fiddling around until it looked reasonable. Later we show another in-line function for measuring the actual width of something.

A minus sign means upward motion, while no sign or a plus sign means move down the page. Thus `\v'-1'` means an upward vertical motion of one line space.

There are many other ways to specify the amount of motion. The following three examples are all legal.

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

Notice that the scale specifier (`i`, `p`, or `m`) goes inside the quotes. Any character can be used in place of the quotes; this is also true of all other `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.

### `\h` Function — Arbitrary Horizontal Motion

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

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

causes a backward motion of a tenth of an inch. As a practical matter, consider printing the mathematical symbol '>>'. The standard spacing is too wide, so `eqn` replaces this by

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

to produce ≫.

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

[3] Omar Khayyám — *the Rubáiyát*

```
\w'thing'
```

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

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

As we mentioned above, the default scale factor for all horizontal dimensions is `m` (ems), so here we must have the `u` for machine units, or the motion produced will be far too large. `troff` is quite happy with the nested quotes, by the way, so long as you don't leave any out.

As a live example of this kind of construction, the œ, æ, Œ, and Æ ligatures discussed in the section on ligatures in the chapter *Fonts and Special Characters*, were constructed using the `\h` function to define the following strings:

```
.ds ae a\h'-(\w'a'u*4/10)'e
.ds Ae A\h'-(\w'A'u*4/10)'E
.ds oe o\h'-(\w'o'u*4/10)'e
.ds Oe O\h'-(\w'O'u*4/10)'E
```

and for any given one of those strings, the mess is unscrambled like this:

| *Construct* | *Explanation* |
|---|---|
| `.ds ae` | Define a string called 'ae'. |
| `a` | Letter 'a' in the string. |
| `\h'-(\w'a'u*4/10)'` | Move backward 0.4 of the width of the letter 'a'. |
| `e` | Letter 'e' in the string. |

## 12.3. `\0` Function — Digit-Size Spaces

The in-line `\0` function is an unpaddable white space 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. You could use the digit space to get numerical columns correctly lined up. For example, suppose you have this list of items:

```
.nf
.ta 5n
Step    Description
.sp 5p
1.  Unpack the handy dandy fuse blower.
2.  Inspect for obvious shipping defects.
    .
    .
    .
9.  Find a wall socket.
10. Insert handy dandy fuse blower in wall socket.
11. Push red button to blow all fuses.
.fi
```

When you format this list of operations, you get this result:

```
Step  Description

1.    Unpack the handy dandy fuse blower.
2.    Inspect for obvious shipping defects.
                  .
                  .
                  .
9.    Find a wall socket.
10.   Insert handy dandy fuse blower in wall socket.
11.   Push red button to blow all fuses.
```

As you can see, the numbers do not line up at the decimal point, but instead are lined up on the left. Placing a space character in front of the digits in the input is not sufficient measure to line up the digits at the decimal. A space is not the same width as a digit (at least not in `troff`). A solution is to use the unpaddable digit-space character `\0` in front of the single digits like this:

```
.nf
.ta 5n
Step    \0Description
.sp 5p
\01.    Unpack the handy dandy fuse blower.
\02.    Inspect for obvious shipping defects.
    .
    .
    .
\09.    Find a wall socket.
10. Insert handy dandy fuse blower in wall socket.
11. Push red button to blow all fuses.
.fi
```

Now when you format the text, you get this result:

Step Description

1. Unpack the handy dandy fuse blower.
2. Inspect for obvious shipping defects.

.
.
.

9. Find a wall socket.
10. Insert handy dandy fuse blower in wall socket.
11. Push red button to blow all fuses.

which looks better than the previous example.

## 12.4. '\ ' Function — Unpaddable Space

There is also the in-line \ function, which is the \ character (backslash) followed by a space character. This function is an unpaddable character the width of a space. You can use this to make sure that things don't get split across line boundaries, for instance if you want to see something like **nroff -Tlp** *myfile* in the stream of text, with the command line set off like it was here and ensuring that it all appears on one line, you would type it in as

```
\ \ \f(LBnroff\ -Tlp\fP\ \fImyfile\fP\ \
```

in-line in the text.

## 12.5. \| and \^ Functions — Thick and Thin Spaces

In typography, there are times when you need spaces that are one-sixth or one-twelfth of the width of an em-space. troff supplies the in-line \| function which is one-sixth of an em-space wide — this is sometimes called a 'thick space'. Where would you want such a thing? Well one place it could be used is in making an ellipsis look better. In general, an ellipsis in a proportional font looks too cramped if you just string three dots together:

```
...
```

and the dots tend to look too spread out if you just place spaces between them:

```
. . .
```

and so the answer is often to use the thick space to get a more pleasing effect like this:

```
. . .
```

which was actually achieved by typing:

```
.\|.\|.
```

Lastly, the in-line \^ function is one-twelfth of the width of an em-space space. This function is almost always used for a typographical application called *italic correction*. Consider an italic word followed by some punctuation such as *do tell*! Because the italic letters are slanted to the right, they lean slightly on the trailing punctuation, especially when the last letter is a tall one like the *l* in the example. So, what typographers do is to apply the italic correction in the form of a thin space just before the punctuation, so that the effect is now *do tell*! What we actually typed here was

```
\fIdo tell\fP\^!
```

with the italic correction just before the exclamation mark.

Typing the italic correction at every instance of adjacent Roman and italic text, would be a lot of work. Some macro packages construct special-purpose macros for applying the italic correction. For example, the **–man** macro package has a `.IR` macro that joins alternating italic and Roman words together so that you can italicize parts of words or have italic text with trailing Roman punctuation. You use the `.IR` macro like:

```
.IR well spring
```

to get the composite effect of *well*spring in your text. The `.IR` macro (somewhat simplified) looks like this:

```
.de IR
\&\fI\\$1\^\fR\\$2\fI\\$3\^\fR\\$4\fI\\$5\^\fR\\$6\fI\\$7\^\fR\\$8\fI\\$9\^\fR
..
```

and you can see the italic correction applied after every parameter that is set in the italic font.

## 12.6. `\&` Function — Non-Printing Zero-Width Character

The `\&` function is a character that does not print, and does not take up any space in the output text. You might wonder what use it is at all? One application of the non-printing character used throughout this manual is to display examples of text containing `troff` or `nroff` requests. To print a `troff` request just as it appears in the input, you have to distinguish it from a real `troff` request. You cannot print an example whose input looks just like this:

```
.in +0.5i          indent the text half an inch
     .
     .
     .
lots of lines of text to be processed
     .
     .
     .
.in -0.5i          unindent the text half an inch
```

The `.` characters at the beginning of each line would be interpreted as `troff` requests instead of text representing examples of requests. In such cases, we have to use the `\&` function to stop `troff` or `nroff` from interpreting the `.` at the start of the line as a control character. We would type the example like this:

```
\&.in +0.5i        indent the text half an inch
     \&.
     \&.
     \&.
lots of lines of text to be processed
     \&.
     \&.
     \&.
\&.in -0.5i        unindent the text half an inch
```

Another place where the `\&` function is useful is within some of the other in-line functions such as the `\l` function. The `\l` function draws lines and you type the function like:

```
\l' length   character '
```

where *length* is the length of the line you want to draw, and *character* is the character to use. Sometimes, the *character* might look like a part of *length*, for instance,

```
\l'1.0i='
```

doesn't get you a one-inch line of = signs as you might expect, because the = sign looks like an expression where you are trying to say that "1.0i is equal to" something else. When you encounter this situation, type the `\l` function like this:

```
\l'1.0i\&='
```

and the result is a one-inch line of ========== signs as you see here.

## 12.7. `\o` Function — Overstriking Characters

Automatically-centered overstriking of up to nine characters is possible with the in-line `\o` (overstrike) function. The `\o` function looks like `\o'`*string*`'` where the characters in *string* are overprinted with their centers aligned. This means for example, that you can print from one to nine different characters superimposed upon each other. `troff` determines the width of this "character" you are creating to be the width of the widest character in your string. The superimposed characters are then centered on the widest character. The *string* should *not* contain local vertical motion. The in-line `\o` function is used like this:

```
\o"set of characters"
```

This is useful for printing accents, as in

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

which produces

système téléphonique

The accents are `\(ga` (grave accent) and `\(aa` (acute accent), or `` \` `` and `\´`; remember that each is just one character to `troff`.

```
\o"e\´"
```

produces

é

and

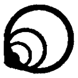
```
\o"\(mo\(sl"
```

produces

∉.

## 12.8. `\z` Function — Zero Motion Characters

You can make your own overstrikes with another special convention, `\z`, the zero-motion command. `\z` *x* 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`, `troff` centers the characters on the widest of them, and there can be no horizontal or vertical motions, so `\z` may be the only way to get what you want:



is produced by

```
.sp 2
\s8\z\(ci\s14\z\(ci\s22\z\(ci\s36\z\(ci
```

The `.sp 2` line is needed to leave enough vertical space for the result.

As another example, an extra-heavy semicolon that looks like

; instead of ; or ;

can be constructed with a big comma and a big period above it:

```
\s+6\z,\v′-0.25m′.\v′0.25m′\s0
```

where `0.25m` is an empirical constant.

As further examples, `\z\(ci\(pl` produces

⊕

and `\(br\z\(rn\(ul\(br` produces the smallest possible constructed box:

□

There is also a more general overstriking function for piling things up vertically — this topic is discussed in the section "`\b` Function — Build Large Brackets" later in this chapter.

## 12.9. `\w` Function — Get Width of a String

Back in the section on using tabs, we saw how we could set tab stops to various positions on the line and lay stuff out in columns based on the tab stops. Sometimes it is hard to figure out where the tab stops should go because you can't always tell in advance how wide things are — this is especially true for proportional fonts (by definition the characters aren't all the same size). Often what you want is to set tab stops based on the width of an item. Then you can set tab stops based on that width and remain independent of the size of the characters if you decide to change point size.

The in-line width function `\w'`*string*`'` generates the numerical width of *string* (in basic units). For example, `.ti -\w'1. 'u` could be used to temporarily indent leftward a distance equal to the size of the string '`1. `'. Size and font changes may be safely embedded in *string*, and do not affect the current environment.

In a previous example we showed how a large capital letter could be placed in a verse with vertical motions and we played some games with indenting to get the thing to come out more-or-less right. The problem with that approach is that we had to measure the size of the character and arrive at the indent by trial and error (actually, error and trial). Another problem is that the measured indent didn't take the point-size into account — if we decide to change sizes, the measurements are all wrong. The width function can measure the size of the thing directly, so here's our example all over again using the `\w` function:

```
.in +\w'\s36A\s0'u
.ti -\w'\s36A\s0'u
\v'1.0'\s36A\s0\v'-1.0'\h'-5p'wake! for Morning in the Bowl of Night
\h'1p'Has flung the Stone that puts the Stars to Flight:
.in -\w'\s36A\s0'u
And Lo! the Hunter of the East has caught
The Sultan's Turret in a Noose of Light.
```

and when we format that text we get this result:

Awake! for Morning in the Bowl of Night
Has flung the Stone that puts the Stars to Flight:
And Lo! the Hunter of the East has caught
The Sultan's Turret in a Noose of Light.

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

Table 12-1 `troff` *Width Function* — `ct` *Number Register Values*

| `ct` Number Register Value | Meaning |
|---|---|
| 0 | all of the characters in *string* were short lower case characters without descenders (like **e**) |
| 1 | at least one character has a descender (like **y**) |
| 2 | at least one character is tall (like **H**) |
| 3 | both tall characters and characters with descenders are present. |

## 12.10. `\k` Function — Mark Current Horizontal Place

The in-line `\k`*x* function stores the current horizontal position in the input line into register *x*. As an example, we could get a bold italic effect by the construction:

```
\kxword \h´|\nxu+2u´word
```

This emboldens *word* by backing up to its absolute (hence, the |) beginning (\kx*word*\h´|\nxu) plus 2 machine units (+2u) and overprinting it, resulting in

***word***

## 12.11. \b Function — Build Large Brackets

The Special (mathematical) font contains a number of characters for constructing large brackets out of pieces. The table below shows the escape-sequences for the individual pieces, what they look like, and their names.

Table 12-2 *Pieces for Constructing Large Brackets*

| *Escape Sequence* | *Character* | *Description* |
|---|---|---|
| `\(lt` | ⎧ | left top of big curly bracket |
| `\(lb` | ⎩ | left bottom of big curly bracket |
| `\(rt` | ⎫ | right top of big curly bracket |
| `\(rb` | ⎭ | right bottom of big curly bracket |
| `\(lk` | ⎨ | left center of big curly bracket |
| `\(rk` | ⎬ | right center of big curly bracket |
| `\(bv` | │ | bold vertical |
| `\(lf` | ⎣ | left floor (left bottom of big square bracket) |
| `\(rf` | ⎦ | right floor (right bottom of big square bracket) |
| `\(lc` | ⎡ | left ceiling (left top of big square bracket) |
| `\(rc` | ⎤ | right ceiling (right top of big square bracket) |

These pieces can be combined into various styles and sizes of brackets and braces by using the in-line \b (for bracketing) function. The \b function is used like this:

```
\b'string '
```

to pile up the characters vertically in *string* with the first character on top and the last on the bottom. The characters are vertically separated by one em and the total pile is centered 1/2-em above the current baseline (1/2-line in nroff). For example:

```
\x' -0.5m' \x'0.5m' \b' \(lc\(lf'E\|\b' \(rc\(rf'
```

produces $\left[\mathrm{E}\right]$ . As with previous examples, we should unscramble the whole mess for you:

| *Escape Sequence* | *Character* | *Description* |
|---|---|---|
| \b | | *start bracketing function* |
| \(lc | ⌈ | *left ceiling* |
| \(lf | ⌊ | *left floor* |
| E | | *letter E* |
| \b | | *start bracketing function* |
| \(rc | ⌉ | *right ceiling* |
| \(rf | ⌋ | *right floor* |

Here's another example of using braces and brackets. You get this effect:

$$\left\{\left[\,x\,\right]\right\}$$

by typing this:

```
\b'\(lt\(lk\(lb' \b'\(lc\(lf' x \b'\(rc\(rf' \b'\(rt\(rk\(rb'
```

## 12.12. \r Function — Reverse Vertical Motions

The \r function makes a single reverse motion of one em upward in troff, and one line upward in nroff.

## 12.13. Drawing Horizontal and Vertical Lines

Typesetting systems commonly have commands to draw horizontal and vertical lines. Of course typographers don't call them lines — they are called 'rules' because once upon a time they were drawn with rulers. troff provides a convenient facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters, and these facilities are described in the subsections following.

### \l Function — Draw Horizontal Lines

The in-line \l (lower-case ell) function draws a horizontal line. For example, the function \l'1.0i' draws a one-inch horizontal line like this ______________ in the text.

The line is actually drawn using the *baseline rule* character in troff, and the underline character in nroff, but you can in fact make the character that draws the line any character you like by placing the character after the length designation. For example, you could draw a two inches of tildes by using \l'2.0i~' to get ˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜ in the text. The construction \L is entirely analogous, except that it draws a vertical line instead of horizontal.

The general form of the \l function is

```
\l 'length character'
```

where *length* is the length of the string of characters to be drawn, and *character* is the character to use to draw the line. If *character* looks like a continuation of *length*, you can insulate *character* from *length* with the zero-width `\&` sequence. If *length* is negative, a backward horizontal motion of size *length* is made *before* drawing the string. Any space resulting from length/(size of character) 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 remainder space is covered by overlapping. If *length* is less than the width of *character*, a single *character* is centered on a distance *length*. As an example, here is a macro to underscore a string:

```
.de us
\\$1\ l´|0\(ul´
..
```

and you use the `.us` macro like this:

```
.us "underlined words"
```

to yield underlined words in the stream of text. You could also write a macro to draw a box around a string:

```
.de bx
\(br\\$1\(br\ l´|0\(rn´\ l´|0\(ul´
..
```

and so you can type:

```
.bx "words in a box"
```

to get some words in a box in the text stream.

### `\L` Function — Draw Vertical Lines

The in-line `\L` (upper-case ell) function draws a vertical line. As in the case of the `\l` function, the general form of the function is

```
\ L 'length character'
```

This draws a vertical line consisting of the (optional) character *character* 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 *length* specifies a line drawn downward and a negative *length* 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.

### Combining the Horizontal and Vertical Line Drawing Functions

The horizontal and vertical line drawing functions may be used in combination to produce large boxes. The zero-width box-rule and the ½-em wide underrule were designed to form corners when using one-em vertical spacings. For example the macro

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

draws a box around some text whose beginning vertical place was saved in number register *z* (using `.mk z`) as done for this paragraph.

### 12.14. `.mc` — Place Characters in the Margin

Many types of documents require placing specific characters in the margins. The most common use of this is placing bars down the margins to indicate what's changed in a document from one revision of a document to the next. This paragraph and the remainder of the text in this section were preceded by a

```
.mc \s12\(br\s0
```

request (that is, place a 12-point box-rule character in the margin) to turn on the marginal bars, and followed by a simple

```
.mc
```

request to turn off the marginal bars.

Currently, this request is not bug-free, and the margin character only appears to the right of the right margin, but not in left margins. Also, you'll notice that the marginal bars do not appear on incomplete lines, such as this one.

### ***Summary of the*** `.mc` ***Request***

| | |
|---|---|
| *Mnemonic:* | margin character |
| *Form of Request:* | `.mc` *c N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Turn off margin characters |
| *Explanation:* | Specifies that a margin character *c* appear a distance *N* to the right of the right margin after each non-empty text line (except those produced by `.tl`). If the output line is too long (as can happen in nofill mode) the character is 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`. |
| *Notes:* | E, m (see Table A-2) |

# 13

# Character Translations

# 13

# Character Translations

## 13.1. Input Character Translations

The newline delimits input lines. In addition, STX, ETX, ENQ, ACK, and BEL are accepted, and may be used as delimiters or translated into a graphic with a `.tr` (translate) request (refer to the section entitled `.tr` — *Output Translation*). All others are ignored.

## 13.2. `.ec` and `.eo` — Set Escape Character or Stop Escapes

The escape character \ introduces *escape sequences* — meaning the following character is something else, or indicates some function. A complete list of such sequences is given in a later chapter. The \ character should not be confused with the ASCII control character ESC of the same name. The escape character can be changed with an `.ec` (escape character) request, and all that has been said about the default \ becomes true for the new escape character. `\e` can be used to print whatever the current escape character is. If necessary or convenient, the escape mechanism can be turned off with an `.eo` (escape off) request and restored with the `.ec` request.

| | ***Summary of the*** `.ec` ***Request*** |
|---|---|
| *Mnemonic:* | escape character |
| *Form of Request:* | `.ec` *c* |
| *Initial Value:* | \ |
| *If No Argument:* | \ |
| *Explanation:* | Set escape character to \, or to *c*, if given. |

| | ***Summary of the*** `.eo` ***Request*** |
|---|---|
| *Mnemonic:* | escape mechanism off |
| *Form of Request:* | `.eo` |
| *Initial Value:* | Escape mechanism is on |
| *If No Argument:* | Turn escape mechanism off. |
| *Explanation:* | Turn escape mechanism off. |

## 13.3. .cc and .c2 — Set Control Characters

Both the control character . and the *no-break* control character ′ may be changed, if desired. Such a change must be compatible with the design of any macros used in the span of the change, and particularly of any trap-invoked macros.

| | ***Summary of the* .cc *Request*** |
|---|---|
| *Mnemonic:* | control character |
| *Form of Request:* | `.cc` *c* |
| *Initial Value:* | . |
| *If No Argument:* | . |
| *Explanation:* | Set the basic control character to *c*, or reset to ' . '. |

| | ***Summary of the* .c2 *Request*** |
|---|---|
| *Mnemonic:* | no-break control character |
| *Form of Request:* | `.c2` *c* |
| *Initial Value:* | ′ |
| *If No Argument:* | ′ |
| *Explanation:* | Set the *no-break* control character to *c*, or reset to ' ′ '. |

## 13.4. .tr — Output Translation

One character can be made a stand-in for another character using the `.tr` (translate) request. All text processing (for instance, character comparisons) takes place with the input (stand-in) character that appears to have the width of the final character. The graphic translation occurs at the moment of output (including diversion).

| | ***Summary of the* .tr *Request*** |
|---|---|
| *Mnemonic:* | translate |
| *Form of Request:* | `.tr` *abcd....* |
| *Initial Value:* | Not Applicable |
| *If No Argument:* | No translation |
| *Explanation:* | Translate *a* into *b*, *c* into *d*, etc. If an odd number of characters is given, the last one is mapped into the space character. To be consistent, a particular translation must stay in effect from *input* to *output* time. |
| *Notes:* | O (see Table A-2) |

# 14

# Automatic Line Numbering

# 14

# Automatic Line Numbering

## 14.1. `.nm` — Number Output Lines

Output lines may be numbered automatically via the `.nm` (number) request. Refer to the following table for a summary of the `.nm` request. When in effect, a three-digit, Arabic number and a digit-space begins each line of output text. The text lines are thus offset by four digit-spaces, and otherwise retain their line length. To keep the right margin aligned with an earlier margin, you may want to reduce the line length by the equivalent of four digit spaces. Blank lines, other vertical spaces, and lines generated by `.tl` are *not* numbered. Numbering can be temporarily suspended with the `.nn` (no number) request (see below), 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).

***Summary of the*** `.nm` ***Request***

| | |
|---|---|
| *Mnemonic:* | numbering |
| *Form of Request:* | `.nm` ± *N M S I* |
| *Initial Value:* | Line numbering turned off. |
| *If No Argument:* | Line numbering turned off. |
| *Explanation:* | Turn on line numbering if ±*N* is given. The next output line numbered is numbered ±*N*. Default values are *M*= 1, *S*= 1, and *I*= 0. *N* is the line number counter (or incrementer if you use ±*N*), *M* is the multiple of the numbered lines to be printed on the page, *S* is the spacing between line numbers and text, and *I* is the amount of indent for the line numbers. 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`. |
| *Notes:* | E (see Table A-2) |

## 14.2. .nn — Stop Numbering Lines

When you are using the .nm request to number lines (as discussed above), you can temporarily suspend the numbering with the .nn (no number) request.

| ***Summary of the*** .nn ***Request*** | |
|---|---|
| *Mnemonic:* | no numbering |
| *Form of Request:* | .nn *N* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | *N*= 1 |
| *Explanation:* | The next *N* text output lines are not numbered. |
| *Notes:* | E (see Table A-2) |

As an example, the paragraph portions of this chapter are numbered with
15 *M*= 3: .nm 1 3 was placed at the beginning of the chapter; .nm was
placed at the end of the first paragraph; and .nm +0 was placed in front of
this paragraph; and .nm finally placed at the end. Line lengths were also
18 changed (by \w'0000'u) to keep the right side aligned.

Another example is

```
.nm +5 5 x 3
```

21 which turns on numbering with the line number of the next line to be 5 greater than the last-numbered line, *M*= 5, spacing *S* is untouched, and with the indent *I* set to 3.

# 15

# Conditional Requests

# 15

# Conditional Requests

## 15.1. `.if` — Conditional Request

Suppose we want the `.SH` macro to leave two extra inches of space just before section 1, but nowhere else. The cleanest way to do that is to test inside the `.SH` macro whether the section number is 1, and add some space if it is. The `.if` request provides the conditional test that we can add 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 — here a request. If the condition is false, or zero, or negative, the rest of the line is skipped.

It is possible to perform more than one request 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 a `.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. `troff` also provides an 'if-else' construction, which we will not go into here.

A condition can be negated by preceding it with `!`; we get the same effect as above (but less clearly) by using

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

There are a handful of other conditions that can be tested with `.if`. For example, is the current page even or odd?

```
.if e .tl ''even page title''
.if o .tl ''odd page title''
```

gives facing pages different titles when used inside an appropriate new page macro.

Two other conditions are `t` and `n`, which tell you whether the formatter is `troff` or `nroff`.

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

Finally, string comparisons may be made in an `.if`:

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

does 'stuff' if *string1* is the same as *string2*. The character separating the strings can be anything reasonable that is not contained in either string. The strings themselves can reference strings with `\*`, arguments with `\$`, and so on.

In the following table, *c* is a one-character, built-in condition name, ! signifies *not*, *N* is a numerical expression, *string1* and *string2* are strings delimited by any non-blank, non-numeric character not in the strings, and *anything* represents what is conditionally accepted.

***Summary of the* `.if` *Requests***

| | |
|---|---|
| *Mnemonic:/if, if-else, else* | |
| *Form of Request:* | `.if` *c anything* |
| *Initial Value:* | Not Applicable |
| *If No Argument:* | Not Applicable |
| *Explanation* | If condition *c* true, accept *anything* as input. In multi-line case use *\{anything\}.* |
| *Form of Request:* | `.if !`*c anything* |
| *Explanation* | If condition *c* false, accept *anything.* |
| *Form of Request:* | `.if` *N anything* |
| *Explanation* | If expression *N* > 0, accept *anything.* |
| *Form of Request:* | `.if !`*N anything* |
| *Explanation* | If expression *N* ≤ 0, accept *anything.* |
| *Form of Request:* | `.if` *' string1 'string2 ' anything* |
| *Explanation* | If *string1* identical to *string2*, accept *anything.* |
| *Form of Request:* | `.if !`*' string1 'string2 ' anything* |
| *Explanation* | If *string1* is not identical to *string2*, accept *anything.* |
| *Form of Request:* | `.ie` *c anything* |
| *Explanation* | If portion of if-else (like above `if` forms). |
| *Form of Request:* | `.el` *anything* |
| *Explanation* | Else portion of if-else. |

The built-in condition names are:

Table 15-1 *Built-In Condition Names for Conditional Processing*

| *Condition Name* | *True If* |
|---|---|
| **o** | Current page number is odd |
| **e** | Current page number is even |
| **t** | Formatter is `troff` |
| **n** | Formatter is `nroff` |

If the 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), *anything* is accepted as input. If a ! precedes the condition, number, or string comparison, the sense of the acceptance is reversed.

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

### 15.2. `.ie` and `.el` — If-Else and Else Conditionals

The request `.ie` (if-else) is almost 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. Refer to the *Summary of the* `.if` *Requests* for summaries of `.ie` and `.el`.

Some examples are:

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

which outputs a title if the page number is even; and

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

which treats page 1 differently from other pages.

### 15.3. `.ig` — Ignore Input Text

Another mechanism for conditionally accepting input text is via the `.ig` (ignore) request. Basically, you place the `.ig` request before a block of text you want to ignore:

```
.ig        start of ignored block of text
           .
           .
           .
block of text you don't want to appear in the printed output
           .
           .
           .
. .        end of ignore block signalled with . .
```

The `.ig` request functions like a macro definition via the `.de` request except that the text between the `.ig` and the terminating `. .` is discarded instead of being processed for printing.

You can give the `.ig` request an argument — that is, an

```
    .ig xy
```

request ignores all text up to and including a line that reads

```
    .xy
```

which looks just like a request:

```
.ig ZZ        start of ignored block of text
          .
          .
          .
block of text you don't want to appear in the printed output
          .
          .
          .
.ZZ       end of ignore block signalled with  .ZZ
```

You can of course combine the `.ig` request with the other conditionals to ignore a block of text if a condition is satisfied. For example, you might want to omit blocks of text if the printed pages are destined for different audiences:

```
.nr W 1             This manual is for Wizards only
          .
          .
          .
    further processing
          .
          .
          .
.if \nW .ig WZ          If the manual is for wizards
          .
          .
          .
Tutorial material beneath the attention of wizards
          .
          .
          .
.WZ           end of ignored block of text
```

### *Summary of the* `.ig` *Request*

| | |
|---|---|
| *Mnemonic:* | ignore |
| *Form of Request:* | `.ig` *yy* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | Ignore text up to a line starting with `..` |
| *Explanation:* | Ignore input lines up to and including a line starting with `.`*yy* — use `..` if no argument is specified on the request. `.ig` behaves exactly like the `.de` (define macro) request except that the input is discarded. The input is read in copy mode, and any auto-incremented number registers will be affected. |

# 16

# Debugging Requests

# 16

# Debugging Requests

`troff` and `nroff` resemble languages for programming a typesetter rather than a mechanism to describe how a document should be put together. There are times when you just can't figure out why things are going wrong and not generating results as advertised. The requests described here are for dyed-in-the-wool macro wizards.

## 16.1. `.pm` — Display Names and Sizes of Defined Macros

The `.pm` (print macros) request displays the names of all defined macros and how big they are. Why would anybody want to do such a thing? Well, if you're using a macro as a diversion, you might find out (by printing its size) that it is far bigger than you expect (that it's swallowing your entire file).

| | ***Summary of the*** `.pm` ***Request*** |
|---|---|
| *Mnemonic:* | print macros |
| *Form of Request:* | `.pm` *t* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | All |
| *Explanation:* | Print 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. |

## 16.2. .fl — Flush Output Buffer

The .fl (flush) request flushes the output buffer — this can be used when you're using nroff interactively.

| | *Summary of the* `.fl` *Request* |
|---|---|
| *Mnemonic:* | flush |
| *Form of Request:* | `.fl` |
| *Initial Value:* | Not applicable |
| *If No Argument:* | adjusting is turned off |
| *Explanation:* | Flush output buffer. Used in interactive debugging to force output. |

## 16.3. .ab — Abort

A final useful request in the debugging category is the .ab (abort) request which basically bails out and stops the formatting.

| | *Summary of the* `.ab` *Request* |
|---|---|
| *Mnemonic:* | abort |
| *Form of Request:* | `.ab` *text* |
| *Initial Value:* | Not applicable |
| *If No Argument:* | No text is displayed |
| *Explanation:* | Displays *text* and terminates without further processing. If *text* is missing, 'User Abort' is displayed. Does not cause a break. The output buffer is flushed. |

# 17

# Environments

# 17

# Environments

As we mentioned, there is a potential problem when going across a page boundary: parameters like size and font for a page title may well be different from those in effect in the text when the page boundary occurs. `troff` provides a very general way to deal with this and similar situations. There are six environments, each of which has independently-settable versions of many of the parameters associated with processing, including size, font, line and title lengths, fill/nofill mode, tab stops, and even partially-collected lines. Thus the titling problem may be readily solved by processing the main text in one environment and titles in a separate one with its own suitable parameters.

## 17.1. `.ev` — Switch Environment

The command `.ev` *n* shifts to environment *n*; *n* must be in the range 0 through 2. A `.ev` command with no argument returns to the previous environment. Environment names are maintained in a stack, so calls for different environments may be nested and unwound consistently.

When `troff` starts up, environment 0 is the default environment, so in general, the main text of your document is processed in this environment in the absence of any information to the contrary. Given this, we can modify the `.NP` (new page) macro 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 and is thus easier to understand and change.

Another major application for environments is for blocks of text that must be kept together.

A number of the parameters that control the text processing are gathered together into an environment, which can be switched by the user. The environment parameters are those associated with requests noting E in their *Notes* column; in addition, partially-collected lines and words are in the environment. Everything else is global; examples are page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions. All environments are initialized with default parameter values.

| ***Summary of the* `.ev` *Request*** | |
|---|---|
| *Mnemonic:* | environment |
| *Form of Request:* | `.ev` *N* |
| *Initial Value:* | *N*=0 |
| *If No Argument:* | Switch back to previous environment |
| *Explanation:* | Switch to environment *N*, where 0≤*N*≤2. Switching is done in push-down fashion so that restoring a previous environment must be done with `.ev` rather than specific reference. |

# A

# troff Request Summary

# A

# `troff` Request Summary

This appendix is a quick-reference summary of `troff` and `nroff` requests. In the following table, values separated by a `:` are for `nroff` and `troff` respectively.

The notes in column four are explained at the end of this summary.

Table A-1 *Summary of* `nroff` *and* `troff` *Requests*

| *Request Form* | *Initial Value* | *If No Argument* | *Notes* | *Explanation* |
|---|---|---|---|---|
| **.ab** *text* | none | User Abort | — | Displays *text* and terminates without further processing; flush output buffer. |
| **.ad** *c* | adj,both | adjust | E | Adjust output lines with mode *c* from `. j`. |
| **.af** *R c* | Arabic | — | — | Assign format to register *R* (*c* = `1`, `i`, `I`, `a`, `A`). |
| **.am** *xx yy* | — | .*yy*=.. | — | Append to a macro. |
| **.as** *xx string* | — | ignored | — | Append *string* to string *xx*. |
| **.bd** *F N* | off | — | P | Embolden font *F* by *N*–1 units.† |
| **.bd S** *F N* | off | — | P | Embolden Special Font when current font is *F*.† |
| **.bp** ±*N* | N=1 | — | B‡,v | Eject current page. Next page is number *N*. |
| **.br** | — | — | B | Break. |
| **.c2** *c* | ´ | ´ | E | Set nobreak control character to *c*. |
| **.cc** *c* | . | . | E | Set control character to *c*. |

Table A-1 *Summary of* `nroff` *and* `troff` *Requests— Continued*

| *Request Form* | *Initial Value* | *If No Argument* | *Notes* | *Explanation* |
|---|---|---|---|---|
| **`.ce`** *N* | off | *N*=1 | B,E | Center following *N* input text lines. |
| **`.ch`** *xx N* | — | — | v | Change trap location. |
| **`.cs`** *F N M* | off | — | P | Constant character space (width) mode (font *F* ).† |
| **`.cu`** *N* | off | *N*=1 | E | Continuous underline in `nroff`; like `.ul` in `troff`. |
| **`.da`** *xx* | — | end | D | Divert and append to *xx*. |
| **`.de`** *xx yy* | — | .*yy*=.. | — | Define or redefine macro *xx; end* at call of *yy*. |
| **`.di`** *xx* | — | end | D | Divert output to macro *xx*. |
| **`.ds`** *xx string* | — | ignored | — | Define a string *xx* containing *string*. |
| **`.dt`** *N xx* | — | off | D,v | Set a diversion trap. |
| **`.ec`** *c* | \ | \ | — | Set escape character. |
| **`.el`** *anything* | — | — | — | Else portion of if-else. |
| **`.em`** *xx* | none | none | — | End macro is *xx*. |
| **`.eo`** | on | — | — | Turn off escape character mechanism. |
| **`.ev`** *N* | N=0 | previous | — | Environment switched (*push down*). |
| **`.ex`** | — | — | — | Exit from `nroff`/`troff`. |
| **`.fc`** *a b* | off | off | — | Set field delimiter *a* and pad character *b*. |
| **`.fi`** | fill | — | B,E | Fill output lines. |
| **`.fl`** | — | — | B | Flush output buffer. |
| **`.fp`** *N F* | R,I,B,S | ignored | — | Font named *F* mounted on physical position 1≤*N*≤4. |

Table A-1 *Summary of* `nroff` *and* `troff` *Requests— Continued*

| *Request Form* | *Initial Value* | *If No Argument* | *Notes* | *Explanation* |
|---|---|---|---|---|
| **.ft** *F* | Roman | previous | E | Change to font *F* = *x*, *xx*, or 1 through 4. Also \f*x*, \f(*xx*, \f*N*. |
| **.fz** *S F N* | none | — | — | Forces font *F* or *S* for special characters to be in size *N*. |
| **.hc** *c* | \% | \% | E | Hyphenation indicator character *c*. |
| **.hw** *word1 ...* | ignored | — | — | Exception words. |
| **.hy** *N* | on | previous | E | Hyphenate. *N* = mode. |
| **.ie** *c anything* | — | — | — | If portion of if-else; all above forms (like `.if`). |
| **.if** *c anything* | — | — | — | If condition *c* true, accept *anything* as input, for multi-line use \{*anything* \}. |
| **.if** !*c anything* | — | — | — | If condition *c* false, accept *anything*. |
| **.if** *N anything* | — | — | — | If expression *N* > 0, accept *anything*. |
| **.if** !*N anything* | — | — | — | If expression *N* ≤ 0, accept *anything*. |
| **.if** *´string1 ´string2 ´ anything* | — | — | — | If *string1* identical to *string2*, accept *anything*. |
| **.if** !*´string1 ´string2 ´ anything* | — | — | — | If *string1* not identical to *string2*, accept *anything*. |
| **.ig** *yy* | — | .*yy*=.. | — | Ignore until call of *yy*. |
| **.in** ±*N* | N=0 | previous | B,E,m | Indent. |
| **.it** *N xx* | — | off | E | Set an input-line count trap. |
| **.lc** *c* | . | none | E | Leader repetition character. |
| **.lg** *N* | on | on | — | Ligature mode on if *N*>0. |
| **.ll** ±*N* | 6.5 in | previous | E,m | Line length. |
| **.ls** *N* | N=1 | previous | E | Output *N*–1 *V*s after each text output line. |

Table A-1 *Summary of* `nroff` *and* `troff` *Requests— Continued*

| *Request Form* | *Initial Value* | *If No Argument* | *Notes* | *Explanation* |
|---|---|---|---|---|
| `.lt` ±*N* | 6.5 in | previous | E,m | Length of title. |
| `.mc` *c N* | — | off | E,m | Set margin character *c* and separation *N*. |
| `.mk` *R* | none | internal | D | Mark current vertical place in register *R*. |
| `.na` | adjust | — | E | No output line adjusting. |
| `.ne` *N* | — | *N*=1*V* | D,v | Need *N* vertical space (*V* = vertical spacing). |
| `.nf` | fill | — | B,E | No filling or adjusting of output lines. |
| `.nh` | hyphenate | — | E | No hyphenation. |
| `.nm` ±*N M S I* | off | — | E | Number mode on or off, set parameters. |
| `.nn` *N* | — | *N*=1 | E | Do not number next *N* lines. |
| `.nr` *R* ±*N M* | — | — | u | Define and set number register *R* by ±*N*; auto-increment by *M*. |
| `.ns` | space | — | D | Turn no-space mode on. |
| `.nx` *filename* | — | end-of-file | — | Next file. |
| `.os` | — | — | — | Output saved vertical distance. |
| `.pc` *c* | % | off | — | Page number character. |
| `.pi` *program* | — | — | — | Pipe output to *program* (`nroff` only). |
| `.pm` *t* | — | all | — | Print macro names and sizes. If *t* present, print only total of sizes. |
| `.ps` ±*N* | 10-point | previous | E | Point size, also \s±*N*.† |
| `.pl` ±*N* | 11 in | 11 in | v | Page length. |
| `.pn` ±*N* | N=1 | ignored | — | Next page number is *N*. |
| `.po` ±*N* | 0: 26/27 in | previous | v | Page offset. |

Table A-1 *Summary of* `nroff` *and* `troff` *Requests— Continued*

| *Request Form* | *Initial Value* | *If No Argument* | *Notes* | *Explanation* |
|---|---|---|---|---|
| `.rd` *prompt* | — | *prompt*=BEL | — | Read insertion. |
| `.rn` *xx yy* | — | ignored | — | Rename request, macro, or string *xx* to *yy*. |
| `.rm` *xx* | — | ignored | — | Remove request, macro, or string. |
| `.rr` *R* | — | — | — | Remove register *R*. |
| `.rs` | — | — | D | Restore spacing. Turn no-space mode off. |
| `.rt` ±*N* | none | internal | D,v | Return *(upward only)* to marked vertical place. |
| `.so` *filename* | — | — | — | Interpolate contents of source file *name* when `.so` encountered. |
| `.sp` *N* | — | *N*=1*V* | B,v | Space vertical distance *N in either direction.* |
| `.ss` *N* | 12/36 em | ignored | E | Space-character size set to *N*/36 em.† |
| `.sv` *N* | — | *N*=1*V* | v | Save vertical distance *N*. |
| `.ta` *Nt* ... | 0.8: 0.5in | none | E,m | Tab settings: *left* type, unless *t* equals `R` (right), or `C` (centered). |
| `.tc` *c* | space | removed | E | Tab repetition character. |
| `.ti` ±*N* | — | ignored | B,E,m | Temporary indent. |
| `.tl` *'left'center'right'* | — | — | | Three-part title. |
| `.tm` *string* | — | newline | — | Print *string* on terminal (to standard error). |
| `.tr` *abcd....* | none | — | O | Translate *a* into *b*, *c* into *d*, etc. on output. |
| `.uf` *F* | Italic | Italic | — | Underline font set to *F* (to be switched to by `.ul`). |
| `.ul` *N* | off | *N*=1 | E | Underline *N* input lines (italicize in `troff`). |

Table A-1 *Summary of* nroff *and* troff *Requests— Continued*

| *Request Form* | *Initial Value* | *If No Argument* | *Notes* | *Explanation* |
|---|---|---|---|---|
| **.vs** *N* | 1/6in:12pts | previous | E,p | Vertical base line spacing (*V*). |
| **.wh** *N xx* | — | — | v | Set location trap. Negative is with respect to page bottom. |

† Point size changes have no effect in nroff.

‡ The use of ´ as the control character (instead of .) suppresses the break function.

Table A-2 *Notes in the Tables*

| *Note* | *Explanation* |
|---|---|
| B | Request normally causes a break. |
| D | Mode or relevant parameters associated with current diversion level. |
| E | Relevant parameters are a part of the current environment. |
| O | Must stay in effect until logical output. |
| P | Mode must be still or again in effect at the time of physical output. |
| v | Default scale indicator — if not specified, scale indicators are *ignored.* |
| p | Default scale indicator — if not specified, scale indicators are *ignored.* |
| m | Default scale indicator — if not specified, scale indicators are *ignored.* |
| u | Default scale indicator — if not specified, scale indicators are *ignored.* |

# B

# Font and Character Examples

# B

# Font and Character Examples

## B.1. Font Style Examples

The following fonts are printed in 12-point, with a vertical spacing of 14-point, and with non-alphanumeric characters separated by ¼-em space. They are Times Roman, Italic, Bold, and a special mathematical font.

Times Roman

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
● □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢ ® © ™

*Times Italic*

*abcdefghijklmnopqrstuvwxyz*
*ABCDEFGHIJKLMNOPQRSTUVWXYZ*
*1234567890*
*! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |*
*● □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢ ® © ™*

**Times Bold**

**abcdefghijklmnopqrstuvwxyz**
**ABCDEFGHIJKLMNOPQRSTUVWXYZ**
**1234567890**
**! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |**
**● □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢ ® © ™**

Special Mathematical Font

" ´ \ ^ _ ` ˜ / < > { } # @ + − = *
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ ς τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ‾ ≥ ≤ ≡ ~ ≈ ≠ → ← ↑ ↓ × ÷ ± ∪ ∩ ⊂ ⊃ ⊆ ⊇ ∞ ∂
§ ∇ ¬ ∫ ∝ ∅ ∈ ‡ ⇒ ⇐ | ○ ⎧ ⎩ ⎫ ⎭ ⎨ ⎬ | ⌊ ⌋ ⌈ ⌉ |

## B.2. Non-ASCII Characters and *minus* on the Standard Fonts

| *Char* | *Input Name* | *Character Name* | *Char* | *Input Name* | *Character Name* |
|---|---|---|---|---|---|
| ’ | ´ | close quote | fi | `\(fi` | fi |
| ‘ | ` | open quote | fl | `\(fl` | fl |
| — | `\(em` | 3/4 Em dash | ff | `\(ff` | ff |
| - | – | hyphen or | ffi | `\(Fi` | ffi |
| - | `\(hy` | hyphen | ffl | `\(Fl` | ffl |
| – | `\-` | current font minus | ° | `\(de` | degree |
| • | `\(bu` | bullet | † | `\(dg` | dagger |
| □ | `\(sq` | square | ′ | `\(fm` | foot mark |
| _ | `\(ru` | rule | ¢ | `\(ct` | cent sign |
| ¼ | `\(14` | 1/4 | ® | `\(rg` | registered |
| ½ | `\(12` | 1/2 | © | `\(co` | copyright |
| ¾ | `\(34` | 3/4 | | | |

## B.3. Non-ASCII Characters and ´, `, _, +, –, =, and * on the Special Font

The ASCII characters @, #, ", ´, `, <, >, \, {, }, ˜, ˆ, and _ exist *only* on the special font and are printed as a 1-em space if that font is not mounted. The following characters exist only on the special font except for the upper case Greek letter names followed by † which are mapped into upper case English letters in whatever font is mounted on font position one (default Times Roman). The special math plus, minus, and equals are provided to insulate the appearance of equations from the choice of standard fonts.

Table B-1 *Summary of* troff *Special Characters*

| *Char* | *Input Name* | *Character Name* | *Char* | *Input Name* | *Character Name* |
|---|---|---|---|---|---|
| + | `\(pl` | math plus | σ | `\(*s` | sigma |
| − | `\(mi` | math minus | ς | `\(ts` | terminal sigma |
| = | `\(eq` | math equals | τ | `\(*t` | tau |
| ∗ | `\(**` | math star | υ | `\(*u` | upsilon |
| § | `\(sc` | section | φ | `\(*f` | phi |
| ´ | `\(aa` | acute accent | χ | `\(*x` | chi |
| ` | `\(ga` | grave accent | ψ | `\(*q` | psi |
| _ | `\(ul` | underrule | ω | `\(*w` | omega |
| / | `\(sl` | slash (matching backslash) | Α | `\(*A` | Alpha† |
| α | `\(*a` | alpha | Β | `\(*B` | Beta† |
| β | `\(*b` | beta | Γ | `\(*G` | Gamma |
| γ | `\(*g` | gamma | Δ | `\(*D` | Delta |
| δ | `\(*d` | delta | Ε | `\(*E` | Epsilon† |
| ε | `\(*e` | epsilon | Ζ | `\(*Z` | Zeta† |
| ζ | `\(*z` | zeta | Η | `\(*Y` | Eta† |
| η | `\(*y` | eta | Θ | `\(*H` | Theta |
| θ | `\(*h` | theta | Ι | `\(*I` | Iota† |
| ι | `\(*i` | iota | Κ | `\(*K` | Kappa† |

Table B-1 *Summary of* `troff` *Special Characters— Continued*

|  | *Input* | *Character* |  | *Input* | *Character* |
|---|---|---|---|---|---|
| ***Char*** | ***Name*** | ***Name*** | ***Char*** | ***Name*** | ***Name*** |
| κ | `\(*k` | kappa | Λ | `\(*L` | Lambda |
| λ | `\(*l` | lambda | Μ | `\(*M` | Mu† |
| μ | `\(*m` | mu | Ν | `\(*N` | Nu† |
| ν | `\(*n` | nu | Ξ | `\(*C` | Xi |
| ξ | `\(*c` | xi | Ο | `\(*O` | Omicron† |
| ο | `\(*o` | omicron | Π | `\(*P` | Pi |
| π | `\(*p` | pi | Ρ | `\(*R` | Rho† |
| ρ | `\(*r` | rho | Σ | `\(*S` | Sigma |
| Τ | `\(*T` | Tau† | ∞ | `\(if` | infinity |
| Υ | `\(*U` | Upsilon | ∂ | `\(pd` | partial derivative |
| Φ | `\(*F` | Phi | ∇ | `\(gr` | gradient |
| Χ | `\(*X` | Chi† | ¬ | `\(no` | not |
| Ψ | `\(*Q` | Psi | ∫ | `\(is` | integral sign |
| Ω | `\(*W` | Omega | ∝ | `\(pt` | proportional to |
| √ | `\(sr` | square root | ∅ | `\(es` | empty set |
| ‾ | `\(rn` | root en extender | ∈ | `\(mo` | member of |
| ≥ | `\(>=` | >= | \| | `\(br` | box vertical rule |
| ≤ | `\(<=` | <= | ‡ | `\(dd` | double dagger |
| ≡ | `\(==` | identically equal | ⇒ | `\(rh` | right hand |
| ≈ | `\(~=` | approx = | ⇐ | `\(lh` | left hand |
| ~ | `\(ap` | approximates | \| | `\(or` | or |
| ≠ | `\(!=` | not equal | ○ | `\(ci` | circle |
| → | `\(->` | right arrow | ⎧ | `\(lt` | left top of big curly bracket |
| ← | `\(<-` | left arrow | ⎩ | `\(lb` | left bottom |
| ↑ | `\(ua` | up arrow | ⎫ | `\(rt` | right top |
| ↓ | `\(da` | down arrow | ⎭ | `\(rb` | right bot |
| × | `\(mu` | multiply | ⎨ | `\(lk` | left center of big curly bracket |
| ÷ | `\(di` | divide | ⎬ | `\(rk` | right center of big curly bracket |
| ± | `\(+-` | plus-minus | \| | `\(bv` | bold vertical |
| ∪ | `\(cu` | cup (union) | ⌊ | `\(lf` | left floor (left bottom of big square bracket) |
| ∩ | `\(ca` | cap (intersection) | ⌋ | `\(rf` | right floor (right bottom) |
| ⊂ | `\(sb` | subset of | ⌈ | `\(lc` | left ceiling (left top) |
| ⊃ | `\(sp` | superset of | ⌉ | `\(rc` | right ceiling (right top) |
| ⊆ | `\(ib` | improper subset | \ | `\e` | backslash (escape character) |
| ⊇ | `\(ip` | improper superset |  |  |  |

# 8

# Manipulating Files

# 8

# Manipulating Files

## 8.1. Comparing Different Files

Occasionally you want to know whether two files are identical, or if they are not, what the differences are. There exist several different text utilities for comparing the contents of files. You can choose the command best for the task at hand, based on what kind of information it conveys to you. Most of the commands issue no output if the files are the same. Some return terse output stating barely more than the fact that the files differ. Others give a more complete summary of how the files differ and how you would have to modify one file to match the other(s).

The command cmp is an example of a command that issues terse output. At most, cmp prints the byte and line number where the files differ. Two other functions for directly comparing files are diff and comm. comm compares two files, putting the comparison information into three different columns: column one lists lines only in *file1*, column two lists lines only in *file2*, and column three lists lines common to both files. diff compares files and also directories. A special version of diff, diff3, also compares three files, identifying the differing contents with special flags.

The relational database operator join compares a specific field or fields in two files. Each time join finds the compared fields in the two files identical, it produces one output line.

For comparing adjacent lines in a single file, there is the command uniq. uniq can be made to report merely the repeated lines or to count them or to remove all but the first occurrence.

### Comparing Binaries with cmp

The command cmp is for comparing two files. The synopsis of the cmp command is:

```
cmp [-l] [-s] file1 file2
```

cmp compares *file1* and *file2*. If *file1* is the standard input ('–'), cmp reads from the standard input. Under default options, cmp makes no comment if the files are the same. If the files differ, cmp announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

The options available with cmp are:

-l Print the byte number (decimal) and the differing bytes (octal) for each difference.

-s Print nothing for differing files; return codes only.

## Comparing Text with diff

For summarizing the differences between two files or directories, diff is the appropriate tool. To use the diff command, you would follow one of these models:

```
diff [-cefh] [-b] file1 file2

diff [-Dstring] [-b] file1 file2

diff [-l] [-r] [-s] [ ] [-Sname] [-cefh] [-b] dir1 dir2
```

diff is a differential file comparator. When run on regular files, and when comparing text files that differ during directory comparison (see the notes below on comparing directories), diff tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, diff finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, either may be given as '–', in which case the standard input is used. If *file1* is a directory, a file in that directory whose file-name is the same as the file-name of *file2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

These lines resemble ed commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward you can see how to convert *file2* into *file1*. As in ed, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these specification lines come all the lines that are affected in the first file flagged by the character '<', then all the lines that are affected in the second file flagged by the '>' character.

If both arguments are directories, diff sorts the contents of the directories by name, and then runs the regular file diff program as described above on text files that are different. Binary files that differ, common subdirectories, and files that appear in only one directory are listed.

### diff — First Form

To produce a script of append (a), change (c), and delete (d) commands for the editor ed, which will recreate *file2* from *file1*, use the first form of diff with the option –e.

Extra commands are added to the output when comparing directories with diff –e, so that the result is a Bourne shell (sh) script for converting text files common to the two directories from their state in *dir1* to their state in *dir2*.

To produce a script similar to that using –e, but in the opposite order, that is, to recreate *file1* from *file2*, use diff –f. The script generated with the –f option is not useful with ed, however.

To surround the specification lines the simplest use of diff puts out with some lines of context, use diff –c. The default is to present three lines of context. To change this (to 10, for example), add 10 to the –c option (-c10). With the –c option, the output format is slightly different from other diff output. It begins by identifying the files involved and the dates they were created. Then each change is separated by a line with a dozen stars (*). The lines removed from *file1* are marked with '–'; those added to *file2* are marked '+'. Lines that are changed from one file to the other are marked in both files with '!'.

If you know you've only made small changes to the files you are comparing, and you want to speed up the time diff takes to work, you can use diff –h. This command only does a fast, half-hearted job. diff –h works only when changed stretches are short and well-separated, but does work on files of unlimited length.

Except for the –b option, which my be given with any of the others, the options –c, –e, –f, and –h are mutually exclusive.

### diff — Second Form

To create a merged version of *file1* and *file2* on the standard output with C preprocessor controls included, use the second form of diff with the option -D*string*. Compiling the result without defining *string* is equivalent to compiling *file1*, while compiling the result with *string* defined will yield *file2*.

If you want diff to ignore trailing blanks (spaces and tabs), use the option –b. Other strings of blanks compare equal. The way diff works, when it compares directories with the –b option specified, diff first compares the files (as in cmp), and then decides to run the diff algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical, because the only differences are insignificant blank string differences.

### diff — Third Form

When comparing directories, you might be interested in several different things. If diff puts out a lot of output, you probably want to use the –l option (for long output). Each text file diff is piped through the program pr to paginate it, (see "Printing Files" later in this manual). Other differences are remembered and summarized after all text file differences are reported.

To compare directories and subdirectories, use the –r option. –r applies diff recursively to common subdirectories encountered.

Since diff ordinarily only outputs information on files and directories that differ, if a file or several files are identical in directories you are comparing, you won't see the identical files listed in the output. The –s option reports files that are the same, in addition to the usual `diff` output, which are otherwise not mentioned.

Here are two directories, *macros* and *new*. For this example, here are lists of their contents.

```
hostname% ls macros
Makefile              making.index.msun     summary.msun
SunMacros.msun        mechanisms.msun       test.tr
contents.pic          mmemo.7               text.effects.msun
contentsfile.msun     model.makefile.msun   troff.msun
document.styles.msun  process.pic
intro.msun            structures.msun
```

```
hostname% ls new
Makefile              making.index.msun     summary.msun
SunMacros.msun        mechanisms.msun       test.tr
contents.pic          mmemo.7               text.effects.msun
contentsfile.msun     model.makefile.msun   troff.msun
document.styles.msun  process.pic
intro.msun            structures.msun
```

Right now these two directories are identical. The output of `diff` for these two directories *macros* and *new*, if there are no differences is:

```
hostname% diff macros new
```

The normal output is nothing, no response. Now if we edit some files and remove some others in the directory *new*, leaving the files like this:

```
hostname% ls macros new
macros:
Makefile              making.index.msun     summary.msun
SunMacros.msun        mechanisms.msun       test.tr
contents.pic          mmemo.7               text.effects.msun
contentsfile.msun     model.makefile.msun   troff.msun
document.styles.msun  process.pic
intro.msun            structures.msun

new:
Makefile              intro.msun            structures.msun
SunMacros.msun        making.index.msun     summary.msun
contents.pic          mechanisms.msun       text.effects.msun
document.styles.msun  model.makefile.msun   troff.msun
```

The regular diff output looks like this:

```
hostname% diff macros new
diff macros/Makefile new/Makefile
7c7
< FORMATTER = /usr/local/iroff
---
> FORMATTER = /usr/doctools/bin/troff
Only in macros: contentsfile.msun
diff macros/intro.msun new/intro.msun
0a1
> .LP
6,10c7,9
< Document preparation at Sun Microsystems relies on variations of the
< .I troff
< text formatter as the underlying mechanism for turning your wishes into
< printed words and outlines on paper.  Using
< .I troff
---
> Document preparation at Sun Microsystems relies on variations of the
> troff text formatter as the underlying mechanism for turning your wishes
> into printed words and outlines on paper.  Using troff
Only in macros: mmemo.7
diff macros/model.makefile.msun new/model.makefile.msun
3,7c3
< The
< .I Makefile
< below is the
< .I Makefile
< used to actually make this document:
---
> The Makefile below is the Makefile used to actually make this document:
Only in macros: process.pic
Only in macros: test.tr
hostname%
```

The output of diff is rather cryptic. But if you look carefully at the specification lines and the direction of the angle brackets, you can decipher the results accurately.

To get a more complete picture of how the two directories compare, you might want to know which files are identical and which files exist only in one directory. For this, you use diff -s. The diff -s output from our example above looks like this:

```
hostname% diff -s macros new
diff -s macros/Makefile new/Makefile
7c7
< FORMATTER = /usr/local/iroff
---
> FORMATTER = /usr/doctools/bin/troff
Files macros/SunMacros.msun and new/SunMacros.msun are identical
Files macros/contents.pic and new/contents.pic are identical
Only in macros: contentsfile.msun
Files macros/document.styles.msun and new/document.styles.msun are identical
diff -s macros/intro.msun new/intro.msun
0a1
> .LP
6,10c7,9
< Document preparation at Sun Microsystems relies on variations of the
< .I troff
< text formatter as the underlying mechanism for turning your wishes into
< printed words and outlines on paper.  Using
< .I troff
---
> Document preparation at Sun Microsystems relies on variations of the
> troff text formatter as the underlying mechanism for turning your wishes
> into printed words and outlines on paper.  Using troff
Files macros/making.index.msun and new/making.index.msun are identical
Files macros/mechanisms.msun and new/mechanisms.msun are identical
Only in macros: mmemo.7
diff -s macros/model.makefile.msun new/model.makefile.msun
3,7c3
< The
< .I Makefile
< below is the
< .I Makefile
< used to actually make this document:
---
> The Makefile below is the Makefile used to actually make this document:
Only in macros: process.pic
Files macros/structures.msun and new/structures.msun are identical
Files macros/summary.msun and new/summary.msun are identical
Only in macros: test.tr
Files macros/text.effects.msun and new/text.effects.msun are identical
Files macros/troff.msun and new/troff.msun are identical
hostname%
```

To compare two directories beginning somewhere in the middle of the directories, use the option -S*filename* where *filename* is a file in one of the directories you are comparing. The syntax for this command is

```
diff -Sfilename dir1 dir2
```

For example, comparing the two directories from the example above, and beginning with the file `model.makefile.msun`:

```
hostname% diff -Smodel.makefile.msun macros new
diff macros/model.makefile.msun new/model.makefile.msun
3,7c3
< The
< .I Makefile
< below is the
< .I Makefile
< used to actually make this document:
---
> The Makefile below is the Makefile used to actually make this document:
Only in macros: process.pic
Only in macros: test.tr
```

## Three Files — `diff3`

If you have three versions of a file that you want to compare at once, use the `diff3` command. The synopsis for the `diff3` command is:

```
diff3 [-ex3] file1 file2 file3
```

`diff3` compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

| | |
|---|---|
| `====` | all three files differ |
| `====1` | *file1* is different |
| `====2` | *file2* is different |
| `====3` | *file3* is different |

The type of change required to convert a given range of a given file to a range in some other file is indicated in one of these ways:

```
f : n1 a
```

Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.

```
f : n1 , n2 c
```

Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

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

Under the `-e` option, `diff3` publishes a script for the editor `ed` that will incorporate into *file1* all changes between *file2* and *file3*, (that is, the changes that normally would be flagged ==== and ====3). Option `-x` produces a script to incorporate only changes flagged ====. Option `-3` produces a script to incorporate only changes flagged ====3. The following command will apply the resulting script to *file1*.

```
    (cat script; echo '1,$p') | ed - file1
```

Note: Text lines that consist of a single dot (`.`) will defeat the `-e` option.

Table 8-1 `diff3` *Option Summary*

| *OPTIONS* | |
|---|---|
| `-e` | Publish a script for the editor `ed` that will incorporate into *file1* all changes between *file2* and *file3*, (that is, the changes that normally would be flagged ==== and ====3). |
| `-x` | Produce a script for `ed` to incorporate only changes flagged ====. |
| `-3` | Produce a script for `ed` to incorporate only changes flagged ====3. |

### Finding Common Lines with `comm`

The `comm` command prints lines that are common to two files. `comm` reads *file1* and *file2*, which should be ordered in ASCII collating sequence, but at least in the *same* order, and produces a three-column output:

| **Column 1** | **Column 2** | **Column 3** |
|---|---|---|
| lines only in *file1* | lines only in *file2* | lines in both files |

The synopsis of the `comm` command is:

```
comm [-[123] ] file1 file2
```

As an example of the `comm` command's output, consider these files:

```
hostname% cat all
Aaron
Bruce
Dave
Elaine
Greg
Joe
Jon
Kevin
Larry G
Larry K
Linda
Mary
Mike B
Mike F
Niel
Pam
Randy
Sid
Tad
Tom
Wanda
hostname%
```

```
hostname% cat women
Christy
Cyndi
Elaine
Gale
Jeanette
Julia
Katherine
Katy
Linda
Lori
Mary
Pam
Pat
Patti
Rose Marie
Susan
Wanda
```

Here is the output of comm. The three columns overlap making output from files with long lines a little difficult to read.

```
hostname% comm women all
        Aaron
        Bruce
Christy
Cyndi
        Dave
                Elaine
Gale
        Greg
Jeanette
        Joe
        Jon
Julia
Katherine
Katy
        Kevin
        Larry G
        Larry K
                Linda
Lori
                Mary
        Mike B
        Mike F
        Niel
                Pam
Pat
Patti
        Randy
Rose Marie
        Sid
Susan
        Tad
        Tom
                Wanda
```

The filename '–' means the standard input. The flags 1, 2, or 3, suppress printing of the corresponding column. Thus:

```
hostname% comm -12
```

prints only the lines common to the two files, and

```
hostname% comm -23
```

prints only lines in the first file, but not in the second. (comm -123 does nothing).

## Combining Files with join

To compare two files of database information and output a *join* of two fields, there is a utility called join. join is a relational database operator. The synopsis of the command is:

```
hostname% join [-an] [-e string] [-j[1|2] m] [-o list] [-tc] file1 file2
```

The program join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is '–', the standard input is used.

*file1* and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*. Fields are separated by blanks, tabs or newlines. Multiple separators count as one, and leading separators are discarded.

Note: With default field separation, the collating sequence is that of sort –b. Using the join –t, the sequence is that of a plain sort.

Table 8-2 join Option Summary

| *OPTIONS* | |
|---|---|
| –a*n* | The parameter *n* can be one of the values:<br>1 produce a line for each unpairable line in *file1*.<br>2 produce a line for each unpairable line in *file2*.<br>3 produce a line for each unpairable line in both *file1* and *file2*.<br>in addition to producing the normal output. |
| –e *string* | Replace empty output fields with *string*. |
| –j[1\|2] *m* | Join on the *m*th field of file *n*, where *n* is 1 or 2. If *n* is missing, use the *m*th field in each file. Note that join counts fields from 1 instead of 0 like sort does. |
| –o *list* | Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. |
| –t*c* | Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. |

## Repeated Lines and `uniq`

If you want to check your input file for repeated lines, use `uniq` `uniq` reports repeated lines in a file.

The synopsis of the `uniq` command is:

```
uniq [-udc [+n] [-n] ] [inputfile [outputfile] ]
```

`uniq` reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder of the text (no repeated lines) is written in the output file. Note that repeated lines must be adjacent in order to be found.

Normally, the lines in the input file that were not repeated and the first occurrence of the lines that were repeated forms the output. If you want to isolate either of these functions, you can specify either the `–u` or the `–d` option. `uniq –u` copies only the lines *not* repeated in the original file to the output file. `uniq –d` writes one copy of just the repeated lines to the output file.

In case you are interested in knowing how many occurrences of a given line appear in the input file, you can use the option `uniq –c`. With the `–c` option, you get first the number of occurrences, then the output in default format (all of the unique lines and no adjacent repeated lines).

There is also an option to compare the latter parts of lines rather than entire lines. The *n* arguments specify skipping an initial portion of each line in the comparison:

–n The first *n* fields, together with any blanks before each, are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

Table 8-3 `uniq` *Option Summary*

| ***OPTIONS*** | |
|---|---|
| –u | Copy only those lines that are *not* repeated in the original file. |
| –d | Write one copy of just the repeated lines. |
| –c | Supersedes –u and –d and generates an output report in default style but with each line preceded by a count of the number of times it occurred. |
| –*n* | The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors. |
| +*n* | The first *n* characters are ignored. Fields are skipped before characters. |

## 8.2. Modifying Files

The following commands can be used to modify files: `colrm`, `compact`, `compress`, `fold`, `pack`, `sort`, `split`, `tr`, and `tsort`. All of these commands are described in the *SunOS Reference Manual.*

## 8.3. Printing Files

To print files on paper, use the `lpr` command. To examine the print queue, use the `lpq` command. To remove jobs from the print queue, use the `lprm` command. All three of these commands are described in the *SunOS Reference Manual.*

# Index

# Notes

# Notes

# Notes

# Notes

# Notes

# Notes

# Notes

# C

# Escape Sequences

# C

# Escape Sequences

Note: The escape sequences \\, \ ., \", \$, *, \a, \n, \t, and \(newline) are interpreted in copy mode (see Chapter 10).

Table C-1 troff *Escape Sequences*

| *Escape 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 (dot) (see .de) |
| \ (*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) |
| \& | Non-printing, zero width character |
| \! | Transparent line indicator |
| \" | Beginning of comment |
| \$*N* | Interpolate argument 1≤*N*≤9 |
| \% | Default optional hyphenation character |
| \(*xx* | Character named *xx* |
| **x*, *(*xx* | Interpolate string *x* or *xx* |
| \a | Non-interpreted leader character |
| \b'*abc...*' | Bracket building function |
| \c | Interrupt text processing |
| \d | Forward (down) 1/2-em vertical motion (1/2-line in nroff) |
| \f*x*, \f(*xx*, \f*N* | Change to font named *x* or *xx*, or position *N* |

Table C-1 troff *Escape Sequences— Continued*

| *Escape Sequence* | *Meaning* |
|---|---|
| \h'*N*' | Local horizontal motion; move right *N* (negative=left) |
| \k*x* | Mark horizontal input place in register *x* |
| \l' *Nc*' | Horizontal line drawing function (default character is baseline rule in troff or underline in nroff; optionally with character *c* ) |
| \L'*Nc*' | Vertical line drawing function (default character is box rule; optionally with character *c* ) |
| \n*x*, \n(*xx* | Interpolate number register *x* or *xx* |
| \o' *abc...*' | Overstrike characters *a, b, c, ...* |
| \p | Break and spread output line |
| \r | Reverse one-em vertical motion (reverse line in nroff) |
| \s*N*, \s±*N* | Point-size change function |
| \t | Non-interpreted horizontal tab |
| \u | Reverse (up) 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)* |
| \z*c* | Print *c* with zero width (without spacing) |
| \{ | Begin conditional input |
| \} | End conditional input |
| \(newline) | Concealed (ignored) newline |
| *X* | *X*, any character not listed above |

# D

# Predefined Number Registers

# D

# Predefined Number Registers

Table D-1 *General Number Registers*

| *Register Name* | *Description* |
|---|---|
| c. | Input line-number in current input file; same as .c. |
| % | 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. |

Table D-2 *Read-Only Number Registers*

| *Register Name* | *Description* |
|---|---|
| .$ | Number of arguments available at the current macro level. |
| .A | Set to 1 in troff, if −a option used; always 1 in nroff. |
| .H | Available horizontal resolution in basic units. |
| .L | Current line-spacing parameter (.ls). |
| .P | 1 if current page is printed, otherwise zero. |
| .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*'. |

Table D-2 *Read-Only Number Registers— Continued*

| *Register Name* | *Description* |
|---|---|
| `.c` | Number of *lines* read from current input file. |
| `.d` | Current vertical place in current diversion; equal to `nl`, if no diversion. |
| `.f` | Current font as physical quadrant (1-4). |
| `.h` | Text baseline high-water mark on current page or diversion. |
| `.i` | Current indent. |
| `.j` | Current adjustment mode and type. |
| `.k` | Horizontal text portion size of current output line. |
| `.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 nofill 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 (a string, not a number). |

# E

# troff Output Codes

# E

# `troff` Output Codes

As we mentioned before, `troff` is geared up to produce binary codes for a phototypesetter called a C/A/T. This appendix describes the codes for the C/A/T in detail. This information is for people who want to translate C/A/T codes for other purposes.

The basic mechanism of the C/A/T typesetter is a revolving drum divided into four quadrants. On each quadrant of the drum you can mount a strip of film — one strip of film corresponds to a font. Each font has 108 characters in it. Characters are exposed on the final photographic paper by 'flashing' a light through the appropriate position of the film strip on the drum. The actual font to be used is selected (as you will see later) by a combination of 'rail', 'mag', and 'font-half' — the terms 'rail' and 'mag' are hangovers from very old hot-lead typesetting technology and have no place in electro-mechanical systems, but they were carried over because typesetters can't handle new things. Point size changes are handled in the C/A/T by a series of magnifying lenses.

The C/A/T's basic unit of length (machine unit) is 1/432 inch (there are six of these units to a typesetter's 'point'). The quantum of horizontal motion is one unit. The quantum of vertical motion is three units (1/144 inch or half a point). `troff` uses the same system of units in its internal computations.

The C/A/T phototypesetter is driven by sending it a sequence of one-byte (eight-bit byte) codes to specify characters, fonts, point sizes, and other information. The encoding scheme used was obviously designed by someone wanting to send the minimum amount of information across a communications channel at the expense of doing vast amounts of work in the computer driving the typesetter.

A complete C/A/T file is supposed to start with an *initialize* code (described later), followed by an *escape-16* code, then the body of the text destined for the C/A/T. The whole file ends with 14 inches of trailer, followed by a *stop* code. In practice, looking at `troff`'s output file has generated disagreements on what the file really looks like, but we don't have a C/A/T around to really try it out.

Bit 7 of a code byte classifies the byte into one of two major types:

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Major Code Type | Further Encoding | | | | | | |

The top bit (bit 7) is encoded thus:

**1** — An *Escape Code*, specifying horizontal motion, as described below.

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Bit 7 = 1<br>Escape Code | One's Complement of Amount of Motion | | | | | | |

**0** — indicates that bits 7 and 6 are used to further encode the code byte, as follows:

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Flash Code or<br>Control Code | | Further Encoding | | | | | |

The two upper bits have these meanings:

**00** — A *Flash Code*, which selects a character out of a font, as described below.

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Bits 6 and 7 = 00<br>Flash Code | | Character Number to Flash (1–63) | | | | | |

**01** — A *Control Code*, which is then*further* encoded into one of two categories depending on whether the *next* bit is a one or a zero:

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Control Code | | | Further Encoding | | | | |

**1** — This is a *lead code*, described below, or

**0** — in which case the control code is *further* encoded into one of three categories of:

- □ Initialization and termination.
- □ Selecting fonts.
- □ Specifying the direction of motion for escapes and leading.

We have finally reached the end of this encoding scheme. The following sections discuss each type of code in detail.

## E.1. Codes 00*xxxxxx* — Flash Codes to Expose Characters

A code with the bits six and seven equal to zero (00*xxxxxx*) is a *flash code*. A flash code specifies flashing one of 63 characters — the lower six bits of the flash code specify which character to flash. This is not enough character combinations to select even all the characters within a single font (there are 108 characters per font) and so there are control codes (described later) to select the font and which half of the font. Given that a specific font is selected via the *rail*, *mag*, and (for the eight-font C/A/T) the *tilt* codes, you then select an upper-font-half or a lower-font-half. The lower-font-half is the first 63 characters of the font, and the upper-font-half is the remaining 45 characters of the font. A flash code of greater than 46 in the upper-half of the font is considered illegal.

## E.2. Codes `1`*xxxxxxx* — Escape Codes Specifying Horizontal Motion

A code with bit seven equal to 1 (1xxxxxxx) is an *escape code*. An *escape code* specifies horizontal motion. The C/A/T is a boustrophedonic device — that is, it can move in both directions, and so the direction of motion is specified by one of the control codes described later on. The amount of horizontal motion is specified by the one's complement of the lower seven bits of the escape code.

## E.3. Codes `011`*xxxxx* — Lead Codes Specifying Vertical Motion

A codes with the top three bits equal to `011` is a *lead code*. A *lead code* is a subset of the control codes in that the top three bits are `011`. Such a code specifies vertical motion. The amount of the vertical motion is specified by the one's complement of the lower five bits, in vertical quanta. 'Lead' is a typesetter's term deriving from the days of hot-lead machines — the terminology sticks with us because the industry moves slowly.

## E.4. Codes `0101`*xxxx* — Size Change Codes

A byte with the top four bits equal to `0101` is a *size-change* code. Such a code specifies movement of a lens turret and a doubler lens to change the point size of the characters. The size-change codes are as follows:

Table E-1 *Size Change Codes*

| *Point-Size* | *Binary Code* | *Octal Code* | *Point-Size* | *Binary Code* | *Octal Code* |
|---|---|---|---|---|---|
| 6 | 0101 1000 | 0130 | 16 | 0101 1001 | 0131 |
| 7 | 0101 0000 | 0120 | 18 | 0101 0110 | 0126 |
| 8 | 0101 0001 | 0121 | 20 | 0101 1010 | 0132 |
| 9 | 0101 0111 | 0127 | 22 | 0101 1011 | 0133 |
| 10 | 0101 0010 | 0122 | 24 | 0101 1100 | 0134 |
| 11 | 0101 0011 | 0123 | 28 | 0101 1101 | 0135 |
| 12 | 0101 0100 | 0124 | 36 | 0101 1110 | 0136 |
| 14 | 0101 0101 | 0125 | | | |

Changes in size using the doubler lens change the horizontal position on the page:

| *If you change from:* | *Follow the change with:* |
|---|---|
| Single to double | A forward escape of 55 quanta |
| Double to single | A reverse escape of 55 quanta |

Table E-2 *Single Point-Sizes versus Double Point-Sizes*

| *Single* | *Double* |
|---|---|
| 6 | 16 |
| 7 | 20 |
| 8 | 22 |
| 9 | 24 |
| 10 | 28 |
| 11 | 36 |
| 12 | |
| 14 | |
| 18 | |

## E.5. Codes `0100`*xxxx* — Control Codes

A byte with the top four bits equal to `0100` is a *control code*. Not all of the control codes have meaning to the typesetter. The control codes are in three classes, namely:

- Initialization and termination.
- Selecting fonts.
- Specifying the direction of motion for escapes and leading. The control codes and their meanings are:

Table E-3 *C/A/T Control Codes and their Meanings*

| *Category* | *Meaning* | *Binary Code* | *Octal Code* |
|---|---|---|---|
| Initializing<br>and Terminating | Initialize<br>Stop | 0100 0000<br>0100 1001 | 0100<br>0111 |
| Selecting Fonts | Upper Rail | 0100 0010 | 0102 |
| | Lower Rail | 0100 0001 | 0101 |
| | Upper Mag | 0100 0011 | 0103 |
| | Lower Mag | 0100 0100 | 0104 |
| | Tilt Up | 0100 1110 | 0116 |
| | Tilt Down | 0100 1111 | 0117 |
| | Upper Font Half | 0100 0110 | 0106 |
| | Lower Font Half | 0100 0101 | 0105 |
| Specifying Direction | Escape Forward | 0100 0111 | 0107 |
| | Escape Backward | 0100 1000 | 0110 |
| Of Motion | Lead Forward | 0100 1010 | 0112 |
| | Lead Backward | 0100 1100 | 0114 |

Note that *tilt up* and *tilt down* are *unimplemented op-codes* on the four-font C/A/T. However, the illustrious hackers at Berkeley implemented a program called `rvcat` to drive the Versatec or the Varian printers, and they used the `0116`$_8$ code to mean 'multiply the next lead-code by 64' to avoid having enormous runs of small lead-codes.

## E.6. How Fonts are Selected

Fonts are selected by a combination of *rail*, *mag*, and *tilt*. The *tilt* codes exist only on the eight-font C/A/T and this is the only difference between the two machines that is visible to the user. The standard version of `troff` doesn't know about the eight-font machine — University of Illinois is one of the places that hacked over `troff` to make it understand the eight-font C/A/T. The correspondence between *rail*, *mag*, and *tilt* codes is shown in this table:

Table E-4 *Correspondence Between Rail, Mag, Tilt, and Font Number*

| *Rail* | *Mag* | *Tilt* | *Four-Font* | *Eight-Font* |
|---|---|---|---|---|
| Lower | Lower | Up | 1 | 1 |
| Lower | Lower | Down | 1 | 2 |
| Upper | Lower | Up | 2 | 3 |
| Upper | Lower | Down | 2 | 4 |
| Lower | Upper | Up | 3 | 5 |
| Lower | Upper | Down | 3 | 6 |
| Upper | Upper | Up | 4 | 7 |
| Upper | Upper | Down | 4 | 8 |

## E.7. Initial State of the C/A/T

For those wishing to write postprocessors to hack over C/A/T codes, here is the initial state of the beast:

| *Attribute* | *Initial State* |
|---|---|
| Escape | Forward |
| Lead | Forward |
| Font-Half | Lower |
| Rail | Lower |
| Mag | Lower |
| Tilt | Down |

# Index

sun®
microsystems

# Formatting Documents

Part Number: 800-1756-10
Revision A, of 9 May 1988

Material in this manual comes from a number of sources: *Typing Documents on the UNIX System: Using the –ms Macros with Troff and Nroff,* M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *A Guide to Preparing Documents with –ms,* M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *Document Formatting on UNIX Using the –ms Macros,* Joel Kies, University of California, Berkeley, California; *Tbl — A Program to Format Tables,* M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *A System for Typesetting Mathematics,* Brian W. Kernighan, Lorinda L. Cherry, Bell Laboratories, Murray Hill, New Jersey; *Typesetting Mathematics — User's Guide,* Brian W. Kernighan, Lorinda L. Cherry, Bell Laboratories, Murray Hill, New Jersey; *Updating Publications Lists,* M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *Some Applications of Inverted Indexes on the UNIX System,* M. E. Lesk, Bell Laboratories, Murray Hill, New Jersey; *Writing Papers with Nroff Using –me,* Eric P. Allman, University of California, Berkeley; *The –me Reference Manual,* Eric P. Allman, University of California, Berkeley; and *Introducing the UNIX System,* Henry McGilton, Rachel Morgan, McGraw-Hill Book Company, 1983. These materials are gratefully acknowledged.

# Contents

# Tables

# Figures

# Preface

This manual provides user's guides and reference information for various document processing tools. We assume you are familiar with a terminal keyboard and the Sun system. If you are not, see *Getting Started with SunOS: Beginner's Guide* for information on the basics, like logging in and the Sun file system. If you are not familiar with text editing, read "An Introduction to Text Editing" in the manual *Editing Text Files*, or "An Introduction to Document Preparation" in this manual. Finally, we assume that you are using a Sun Workstation, although specific terminal information is also provided.

If you choose to read one of the user's guides, sit down at your workstation and try the exercises and examples. The reference sections provide additional explanations and examples on how to use certain facilities and can be dipped into as necessary. For additional details on Sun system commands and programs, see the *SunOS Reference Manual.*

## Summary of Contents

This manual is divided into three sections:

- Macro Packages
- `troff` Preprocessors
- Verification and Reformatting Programs

1. *Introduction to Document Preparation* — Describes the basics of text processing, macros and macro packages, provides a guide to the available tools and several simple examples after which to pattern your papers and documents. Newcomers to the Sun document formatters should start here.

In Section I, Macro Packages, the chapters are:

2. *Formatting Documents with the* **-ms** *Macros* — User's guide and reference information for the **-ms** macros for formatting papers and documents. Includes new **-ms** macros.
3. *The* **-man** *Macro Package* — User's guide and reference information for the **-man** macros for formatting manual pages (*man* pages). Includes new options to the **-man** macro package.
4. *Formatting Documents with the* **-me** *Macros* — Describes the **-me** macro package for producing papers and documents.

In Section II, troff Preprocessors, the chapters are:

5. refer — *a Bibliography System* — Explains how to use the bibliographic citation program refer. Includes information on the auxiliary programs addbib, indxbib, lookbib, and sortbib.

6. *Formatting Tables with* tbl — A user's guide and numerous examples to the table processing utility tbl.

7. *Typesetting Mathematics with* eqn — A user's guide to the eqn mathematical equation processor.

Section III, Verification and Formatting Programs, discusses:

8. checknr — a program to report unmatched pairs of macros and unpaired font or size changes.

   spell — a program that prints strings of characters to your terminal screen that spell doesn't have in its dictionary (/usr/dict/words).

   The reformatting commands fmt, deroff, pti, colcrt, col, ul, and ptx.

## Conventions Used in This Manual

Throughout this manual we use

```
hostname%
```

as the prompt to which you type system commands. **Boldface typewriter font** indicates commands that you type in exactly as printed on the page of this manual. Regular typewriter font represents what the system prints out to your screen. Typewriter font also specifies Sun system command names (program names) and illustrates source code listings. *Italics* indicates general arguments or parameters that you should replace with a specific word or string. We also occasionally use italics to emphasize important terms.

# 1

# Introduction to Document Preparation

1

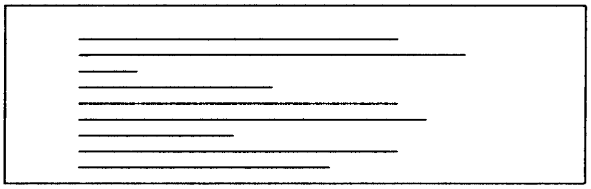
# Introduction to Document Preparation

The document preparation tools `nroff` and `troff` are standard with SunOS. These programs read files containing the text to be formatted, interspersed with requests specifying how output should look. From this, the programs produce formatted output. `nroff` is for typewriter-like printers, while `troff` is for typesetters and laser printers. Although they are separate programs, they are compatible: the formatters share a common command language and produce output from the same input file. Descriptions here apply to both formatters unless stated otherwise.

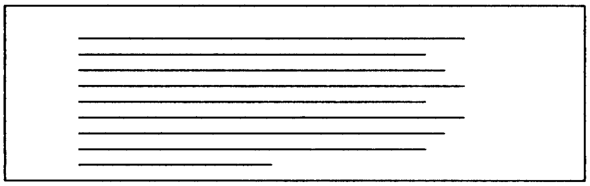
## 1.1. What Do Text Formatters Do?

You can type in text on lines of any length, and the formatters produce lines of uniform length in the finished document. This process is called *filling*, which means that the formatter collects words from what you type as input, and places them on an output line until no more fit within a given line length. The formatter *hyphenates* words automatically, so a line may end with part of a word to produce the right line length. The formatter also *adjusts* a line after it has been filled by inserting spaces between words as necessary to align the right margin exactly.

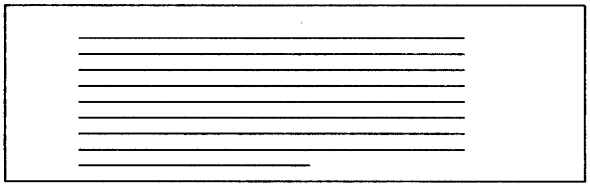
Unfilled text:



Filled but not adjusted:



Filled and adjusted:



Given a file of input consisting only of lines of text without any formatting requests, the formatter simply produces a continuous stream of filled, adjusted and hyphenated output.

To obtain paragraphs, numbered sections, multiple column layout, tops and bottoms of pages, and footnotes, for example, require the addition of formatting requests. Requests look like `.`*xx* where *xx* is one or two lower-case letters or a lower-case letter and a digit. Refer to *Using* `nroff` *and* `troff for details.`

## 1.2. What is a Macro Package?

`nroff` and `troff` provide a flexible, sophisticated command language for requesting operations like those just mentioned. They are very flexible, but this flexibility can make them difficult to use because you have to use several requests to produce a simple format. For this reason, it's a good idea to use a macro package.

A macro is simply a "predefined sequence of `troff` requests or text" which you can use by including just one request in your input file. You can then handle repetitious tasks, such as starting paragraphs and numbering pages, by typing one macro request each time instead of several. For example, some macro requests look like `.`*XX* where *XX* is one or two upper-case letters or an upper-case letter and a digit. (Different macro packages follow various conventions.)

A macro package also does a lot of things without the instructions that you have to give `nroff`, footnotes and page transitions for example. Some packages set up a page layout style by default, but you can change that style if you wish. Although a macro package offers only a limited subset of the wide range of formatting possibilities that `nroff` provides, it is much easier to use. We explain how to use a macro package in conjunction with `nroff` and `troff` in the section "Displaying and Printing Documents."

Sample input with both formatting requests, macros in this case, and text looks like:

```
.LP
Now is the time
for all good men
to come to the aid of their country.
.LP
```

Refer to the chapter "Formatting Documents with the **-ms** Macros" and to the "Quick References" in this chapter for more information on macros.

## 1.3. What is a Preprocessor?

A preprocessor is a program that you run your text file through first before passing it on to a text formatter. You can put tables in a document by preprocessing a file with the table-builder called `tbl`. You can add mathematical equations with their special fonts and symbols with the equation formatters, `eqn` for `troff` files and `neqn` for `nroff` files. These preprocessors convert material entered in their specific command languages to straight `troff` or `nroff` input. Those text formatters then produce the tables or mathematical equations for the output.

What you type in a file is very much the same as for simple formatting. You include table or equation material in your `troff` input file along with ordinary text and add several specific `tbl` or `eqn` requests. Refer to the chapters "Formatting Tables with `tbl`" and "Formatting Mathematics with `eqn`" for details.

## 1.4. Typesetting Jargon

There are several printer's measurement terms that are borrowed from traditional typesetting. These terms describe the size of the letters, the distance between lines and paragraphs, how long each line is, where the text is placed on the page, and so on.

*Point* *Points* specify the *size* of a letter or *type*. A point measures about 1/72 of an inch, which means that there are 72 points to the inch. This manual is in 10-point type, for instance.

*Ems* and *Ens*

*Ems* and *ens* are measures of distance and are proportional to the type size being used. An *em* is the distance equal to the number of points in the width of the letter 'm' in that point size. For examples, here's an em in several point sizes followed by an em dash to show why this is a *proportional* unit of measure. You wouldn't want a 20-point dash if you are printing the rest of a document in 12-point. Here's 12-point:

m
|—|

And here's 20-point:

m
|—|

An *en* space is one half of an *em* or about the width of the letter 'n'. Ens are typically used for indicating indentation.

*Vertical Spacing*

Vertical spacing called *leading* (pronounced 'led-ding') is the distance between the bottom of one line and the bottom of the next. This manual has 12-point vertical spacing for example. The rule of thumb is that the spacing be approximately 20% larger than the character size for easy readability. A printer would call the ratio for this manual "ten on twelve."

*Paragraph Depth*

As there is a specification for the distance between lines, there is also a term for the space between paragraphs. This is the paragraph depth. If you are using the standard `.PP` or `.LP` macro, for instance, the paragraph depth is whatever one vertical space has been set to.

*Paragraph Indent*

This is the amount of space that the first line is indented in relation to the rest of the paragraph. If you use a `.PP` macro to format a standard indented paragraph, the indent is two em-spaces as shown by the first line in this paragraph.

*Line Length*

*Line length* specifies the width of text on a page. Here we use a 5-inch line length. Shortening the line length generally makes text easier to read. Recall that many magazines and newspapers have 2-1/4 inch columns for quick reading.

*Page Offset*

*Page offset* determines the left margin, that is how far in the text is set from the left edge of the paper. On a normal 8-1/2-by-11 letter-size page, the page offset is normally 26/27 of an inch.

*Indent*

The *indent* of text is the distance the text is set in from the page offset. The indent emphasizes the text by setting it off from the rest.

## 1.5. Hints for Typing in Text

The following provides a few tricks for typing in text and for further online editing and formatting.

- A period (.) or apostrophe (') as the first character on a line indicates that the line contains a formatting request. If you type a line of text beginning with either of these *control characters*, `nroff` tries to interpret them as a request, and the rest of the text on that line disappears. If you *have* to type a period or an apostrophe as the first character on a line, escape their normal meanings by prefixing them with a backslash and an ampersand. For instance, to display this sample input:

```
\&.LP
Here is some sample input for a left-blocked paragraph.  In order
to accurately display -ms or troff requests that
begin lines, you have to precede them with the character sequence
backslash, ampersand (\&).  This insulates the macro request
from the beginning of the line so the dot in the first column isn't
seen by troff.
\&.LP
\&.sp
The .LP, .EQ and .EN requests shown here are -ms macro requests
and the .sp line is a typical troff request.
\&.EQ (1.3)
x sup 2 over a sup 2 ~=~ sqrt {p z sup 2 +qz+r}
\&.EN
```

- Following the control character is a one- or two-character *name* of a formatting request. As described earlier, `nroff` and `troff` names usually consist of one or two lower-case letters or a lower-case letter and a digit. **-ms** macro package names usually consist of one or two upper-case letters or one upper-case letter and a digit. For example, `.sp` is an `troff` request for a space and `.PP` is an **-ms** macro request for an indented paragraph.

- End a line of text with the end of a word along with any trailing punctuation. `nroff` inserts a space between whatever ends one line of input text and whatever begins the next.

- Start lines in the input file with something other than a space. A space at the beginning of an input line creates a *break* at that point in the output and `nroff` skips to a new output line, interrupting the process of filling and adjusting. This is the easiest way to get spaces between paragraphs, but it does not leave much flexibility for changing things later.

- Some requests go on a line by themselves, while others can take one or more additional pieces of information on the same line. These extra pieces of information on the request line are called *arguments*. Separate them from the request name and from each other by one or more spaces. Sometimes the argument is a piece of text on which the request operates; other times it can be some additional information about what the request is to do. For example, the vertical space request `.sp 3` shows a `troff` request with one argument. It requests three blank lines.

## 1.6. Types of Paragraphs

There are several types of paragraphs. When should you use one type of paragraph instead of another? Here are a few words about paragraphs, their characteristics, and formatting in general. See the "Types of Paragraphs" figure that follows for examples.

Use regular indented and block paragraphs for narrative descriptions. It is a matter of style as to which type you choose to use. In general, indented paragraphs remove the need for extra space between paragraphs — the indent tells you where the start of the new paragraph is. Most business communication is done with block paragraphs.

If you want to indicate a set of points without any specific order, use a bulleted list. For example:

> There are many kinds of coffee:
>
> - Jamaica Blue Mountain
> - Colombian
> - Java
> - Mocha
> - French Roast
> - Major Dickenson's Blend

When you want to describe a set of things in some order, such as a step-by-step procedure, use a numbered list:

> To repair television, follow these steps:
>
> 1. Remove screws in rear casing.
> 2. Carefully slide out picture tube.
> 3. Gently smash with hammer.

Use description lists to explain a set of related or unrelated things, or sometimes to highlight keywords. For instance,

Options

| | |
|---|---|
| –v | Verbose |
| –f *filename* | Take script from *filename* |
| –o | Use old format |

In typographic parlance, anything that is not part of the "body text" — regular paragraphs and such — is considered a *display*, and often has to be specially handled. Generally a display is "displayed" exactly as you type it or draw it originally, with no interference from the formatter. Displays are used to set off important text, special effects, drawings, or examples, as we do throughout this manual, The following paragraph is a *display*:

> Tom appeared on the sidewalk with a bucket of whitewash and<br>
> a long-handled brush.<br>
> He surveyed the fence, and all gladness left him and<br>
> a deep melancholy settled down upon his spirit.<br>
> Thirty yards of board fence nine feet high.<br>
> Life to him seemed hollow, and<br>
> existence but a burden.

Quotations set off quoted material from the rest of the text for emphasis. For example,

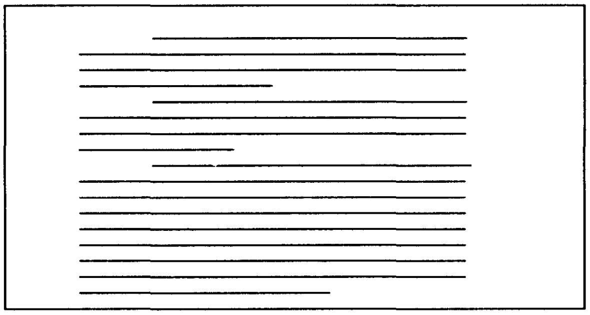
"... in the conversation between Alice and the Queen, we read this piece of homespun philosophy:

> "A slow sort of country!" said the Queen. "Now, *here*, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"
>
> *Through the Looking Glass*
> Lewis Carroll

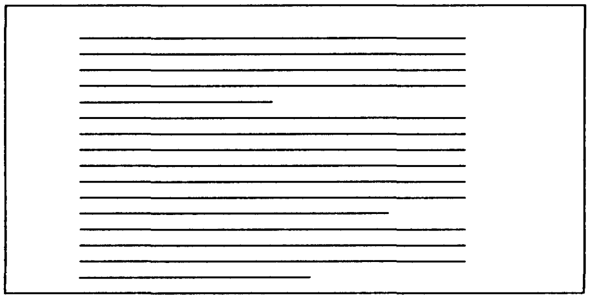
**Paragraph Illustrations**

Examine this section to see how the various paragraph types can serve different functions.

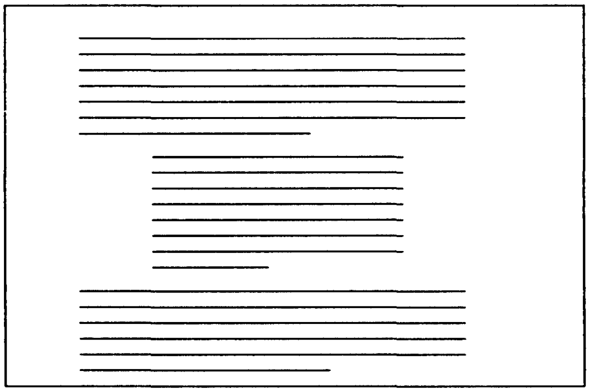
Indented — .PP



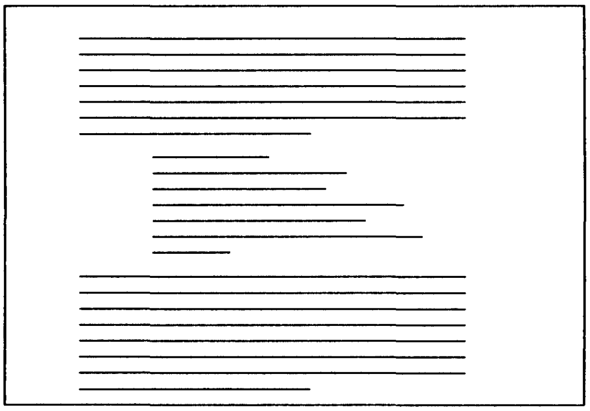
Left Block — .LP



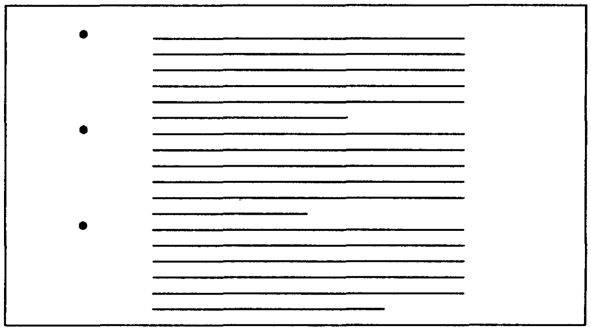
Quotation — .QP



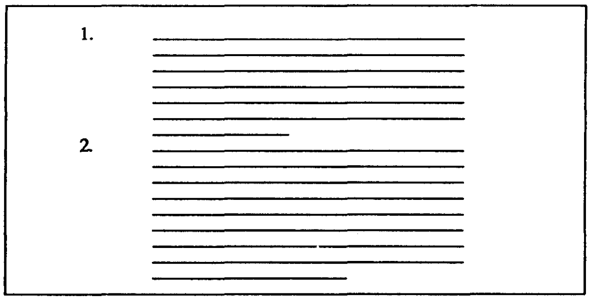
Display — .DS



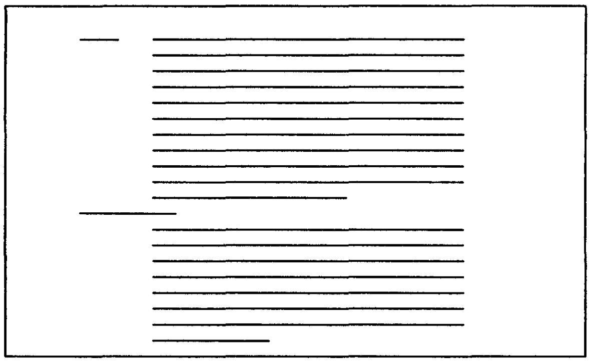
Bulleted — `.IP \(bu`



Numbered — `.IP 1.`



Lists — `.IP` *"tag" n*



## 1.7. Quick References

This section provides some simple templates for producing your documents with the **-ms** macro package.[1] Remember that for a quick, paginated, and justified document, you can simply type an `.LP` to start your document, and then type in the text separated by blank lines to produce paragraphs. Throughout the examples, the text file input is displayed in

```
typewriter font like this
```

while the output is displayed in

Times Roman font.

[1] Some of the material in this section is derived from *A Guide to Preparing Documents with '–ms'*, M.E. Lesk, Bell Laboratories, Murray Hill, New Jersey.

### Displaying and Printing Documents

Use the following to format and print your documents. You can use either `nroff` or `troff` depending on the output you desire. Use `nroff` to either display formatted output on your workstation screen or to print a formatted document. The default is to display on the standard output, your workstation screen. For easy viewing, pipe your output to `more` or redirect the output to a file.

Using `troff` or your installation's equivalent prepares your output for phototypesetting.

Table 1-1 *How to Display and Print Documents*

| ***What You Want to Do*** | ***How to Do It*** |
|---|---|
| Display simple text | **nroff** *–options files* |
| Display text with tables only | **tbl** *files* \| **nroff** *–options* |
| Display text with equations only | **neqn** *files* \| **nroff** *–options* |
| Display text with both tables and equations | **tbl** *files* \| **neqn** \| **nroff** *–options* |
| Print raw text and requests | **pr** *files* \| **lpr** *–Pprinter* |
| Print text | **nroff** *–options files* \| **lpr** *–Pprinter* |
| Print text with tables only | **tbl** *files* \| **nroff** *–options* \| **lpr** *–Pprinter* |
| Print text with equations only | **neqn** *files* \| **nroff** *–options* \| **lpr** *–Pprinter* |
| Print text with both tables and equations | **tbl** *files* \| **neqn** \| **nroff** *–options* \| **lpr** *–Pprinter* |
| Phototypeset simple text | **troff** *–options files* |
| Phototypeset text with tables | **tbl** *files* \| **troff** *–options* |
| Phototypeset text with equations | **eqn** *files* \| **troff** *–options* |
| Phototypeset text with both tables and equations | **tbl** *files* \| **eqn** \| **troff** *–options* |

## Technical Memorandum

Here we provide a sample format for a technical memorandum.

Input:

```
.DA March 11, 1983
.TL
An Analysis of
Cucumbers and Pickles
.AU
A. B. Hacker
.AU
C. D. Wizard
.AI
Stanford University
Stanford, California
.AB
This abstract should be short enough to
fit on a single page cover sheet.
It provides a summary of memorandum
contents.
.AE
.NH
Introduction.
.PP
Now the first paragraph of actual text ...
...
Last line of text.
.NH
References
```

Output:

**An Analysis of**
**Cucumbers and Pickles**

*A. B. Hacker*
*C. D. Wizard*

Stanford University
Stanford, California

*ABSTRACT*

This abstract should be short enough to fit on a single page cover sheet. It provides a summary of memorandum contents.

**1. Introduction.**

Now the first paragraph of actual text ...
Last line of text.

**2. References**

### Section Headings for Documents

```
.NH                         .SH
Introduction.               Appendix I
.PP                         .PP
text text text              text text text
```

1. Introduction  
   text text text

Appendix I  
   text text text

### Changing Fonts

The following table shows the easiest way to change the default roman font to italic or bold. To change the font of a single word, put the word on the same line as the macro request. To change the font in more than one word, put the text on the lines following the macro request.

The font will remain changed until another font change request or a macro request causing a break (a paragraph macro, for example) is encountered.

| Input | Output |
|---|---|
| `.I Hello` | *Hello* |
| `.I`<br>`Prints this line in italics.` | *Prints this line in italics.* |
| `.B Goodbye` | **Goodbye** |
| `.B`<br>`Prints this line in bold.` | **Prints this line in bold.** |
| `.R`<br>`Prints this line in roman.` | Prints this line in roman. |

## Making a Simple List

Use the following template for a simple list.

Input:

```
.IP 1.
J. Pencilpusher and X. Hardwired,
.I
A New Kind of Set Screw,
.R
Proc. IEEE
.B 75
(1976), 23-255.
.IP 2.
H. Nails and R. Irons,
.I
Fasteners for Printed Circuit Boards,
.R
Proc. ASME
.B 23
(1974), 23-24.
.LP          (terminates list)
```

Output:

1. J. Pencilpusher and X. Hardwired, *A New Kind of Set Screw* , Proc. IEEE **75** (1976), 23-255.

2. H. Nails and R. Irons, *Fasteners for Printed Circuit Boards* , Proc. ASME **23** (1974), 23-24.

### Multiple Indents for Lists and Outlines

This template shows how to format lists or outlines.

Input:

```
This is ordinary text to highlight the
results of outline format.
.IP 1.
First level item.
.RS
.IP a)
Second level.
.IP b)
Continued here with another second
level item, but somewhat longer.
.RE
.IP 2.
Return to previous value of the
indenting at this point.
.IP 3.
Another
line.
```

Output:

This is ordinary text to highlight the results of outline format.

1. First level item.

    a) Second level.

    b) Continued here with another second level item, but somewhat longer.

2. Return to previous value of the indenting at this point.

3. Another line.

## Displays

A display does not fill or justify the text. It keeps the text together, and sets the lines off from the rest.

Input:

```
hoboken harrison newark roseville avenue grove street
east orange brick church orange highland avenue
mountain station south orange maplewood millburn short hills
summit new providence
.DS
and now
for something
completely different
.DE
murray hill berkeley heights
gillette stirling millington lyons basking ridge
bernardsville far hills peapack gladstone
```

Output:

hoboken harrison newark roseville avenue grove street east orange brick church orange highland avenue mountain station south orange maplewood millburn short hills summit new providence

> and now
> for something
> completely different

murray hill berkeley heights gillette stirling millington lyons basking ridge bernardsville far hills peapack gladstone

| Display Options | Description |
|---|---|
| `.DS L` | left-adjust |
| `.DS C` | line-by-line center |
| `.DS B` | make block, then center |

## Footnotes

For automatically-numbered footnotes, put the string ** at the end of the text you want to footnote like this:[2]

```
you want to footnote like this:\**
.FS
Here's a numbered footnote.
.FE
```

To mark footnotes with other symbols, put the symbol as the first argument to `.FS` and at the end of the text you want to footnote like this:†

[2] Here's a numbered footnote.

† You can also use an asterisk (*) or a double dagger ‡ (\(dd).

```
you want to footnote like this:\(dg
.FS \(dg
You can also use an asterisk (\fL*\fR)
or a double dagger ‡ (\fL\(dd\fR).
.FE
```

### Keeping Text Together — Keeps

Lines bracketed by the following commands are kept together, and will appear entirely on one page:

```
.KS                                  .KF
    lines of text                        lines of text
    lines of text    not moved           lines of text    may float
    lines of text    through text        lines of text    in text
    lines of text                        lines of text
.KE                                  .KE
```

### Double-Column Format

Put a .2C at the beginning of the material you want printed in two columns. To return to one-column format, use .1C. Note that .1C breaks to a new page.

Input:

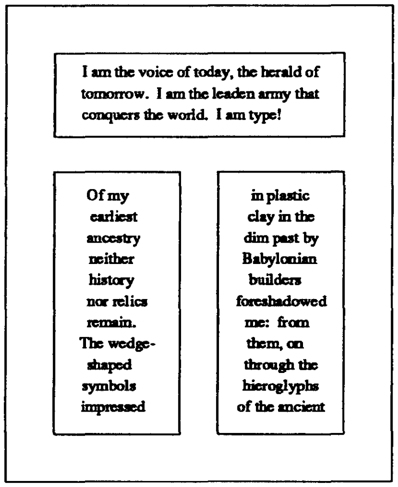
```
.TL
The Declaration of Independence
.sp 2
.2C
.LP
When in the course of human events, it becomes necessary
for one people to dissolve the political bonds which have
connected them with another, and to assume among the
powers of the earth the separate and equal station to which
the laws of Nature and of Nature's God entitle them,
a decent respect to the opinions of . . .
```

Output:

**The Declaration of Independence**

When in the course of human events, it becomes necessary for one people to dissolve the political bonds which have connected them with another, and to assume among the powers of the earth the separate and equal station to which the laws of Nature and of Nature's God entitle them, a decent respect to the opinions of...

The `.2C` macro request only works in this way: When you invoke the `.2C` macro somewhere on a page of text, `.2C` marks that height on the page and produces a narrow column of text all the way to the bottom of that page. When it reaches the bottom of the page, `.2C` resumes the second column at the height it originally marked off when the `.2C` macro was invoked. If the second column is only partially filled up with text when the `.1C` request is encountered, a page break occurs and the single-column text begins the next page. This means `.2C` will do this:

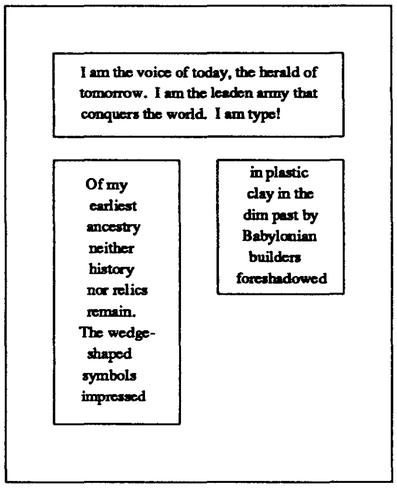


**I am the voice of today, the herald of tomorrow. I am the leaden army that conquers the world. I am type!**

**Of my earliest ancestry neither history nor relics remain. The wedge-shaped symbols impressed**

**in plastic clay in the dim past by Babylonian builders foreshadowed me: from them, on through the hieroglyphs of the ancient**

or this:



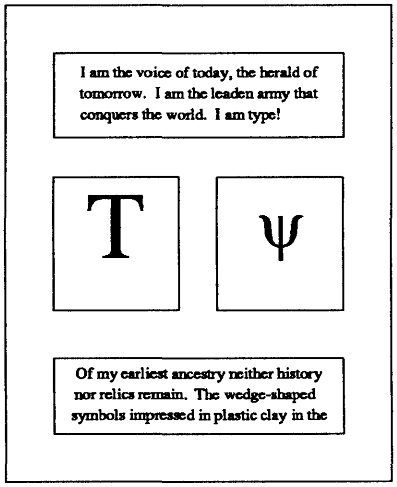
**I am the voice of today, the herald of tomorrow. I am the leaden army that conquers the world. I am type!**

**Of my earliest ancestry neither history nor relics remain. The wedge-shaped symbols impressed**

**in plastic clay in the dim past by Babylonian builders foreshadowed**

The important fact to remember about this macro request, is that it has no other way to determine where to begin column two except upon reaching the bottom of the page.

If the material you want in two columns occupies less space than the distance to the bottom of the current page, it will only occupy one narrow column if you use the `.2C` macro request. This means you cannot do this:



I am the voice of today, the herald of tomorrow. I am the leaden army that conquers the world. I am type!

Of my earliest ancestry neither history nor relics remain. The wedge-shaped symbols impressed in plastic clay in the

## Sample Tables

Two sample table templates follow.

Input:

```
.TS
box center tab (/);
lB lB
l l.
Column Header   Column Header
_
text/text
text/text
text/text
text/text
.TE
```

Output:

| **Column Header** | **Column Header** |
|---|---|
| text | text |
| text | text |
| text | text |
| text | text |

Input:

```
.TS
allbox tab (/);
cB s s
c c c
n n n.
AT&T Common Stock
Year/Price/Dividend
1971/41-54/$2.60
2/41-54/2.70
3/46-55/2.87
4/40-53/3.24
5/45-52/3.40
6/51-59/.95*
.TE
* (first quarter only)
```

Output:

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

* (first quarter only)

The meanings of the key-letters describing the alignment of each entry are:

| `tbl` | **Column** |
|---|---|
| **Key-Letter** | **Described** |
| `c` | centered |
| `r` | right-adjusted |
| `l` | left-adjusted |
| `n` | numerical |
| `a` | alphabetical |
| `s` | spanned |

The global table options are `center`, `expand`, `box`, `doublebox`, `allbox`, `tab` (*x*), and `linesize` (*n*).

Input:

```
.TS
center box tab (/) ;
cB cB
l l.
Name/Definition
_
Gamma/$GAMMA (z) = int sub 0 sup inf t sup {z-1} e sup -t dt$
Sine/$sin (x) = 1 over 2i ( e sup ix - e sup -ix )$
Error/$roman erf (z) = 2 over sqrt pi int sub 0 sup z e sup {-t sup 2} dt$
Bessel/$J sub 0 (z) = 1 over pi int sub 0 sup pi cos ( z sin theta ) d theta$
Zeta/$zeta (s) = sum from k=1 to inf k sup -s ~~( Re~s > 1)$
.TE
```

Output:

| **Name** | **Definition** |
|---|---|
| Gamma | $\Gamma(z)=\int_0^{\infty} t^{z-1}e^{-t}\,dt$ |
| Sine | $\sin(x)=\frac{1}{2i}(e^{ix}-e^{-ix})$ |
| Error | $\mathrm{erf}(z)=\frac{2}{\sqrt{\pi}}\int_0^{z} e^{-t^2}dt$ |
| Bessel | $J_0(z)=\frac{1}{\pi}\int_0^{\pi}\cos(z\sin\theta)d\theta$ |
| Zeta | $\zeta(s)=\sum_{k=1}^{\infty}k^{-s}\quad(\mathrm{Re}\,s>1)$ |

## Writing Mathematical Equations

A displayed equation is marked with an equation number at the right margin by adding an argument to the `.EQ` line:

Input:

```
.EQ (1.3)
x sup 2 over a sup 2 ~=~ sqrt {p z sup 2 +qz+r}
.EN
```

A displayed equation is marked with an equation number at the right margin by adding an argument to the `.EQ` line:

Output:

$$\frac{x^2}{a^2} = \sqrt{pz^2+qz+r} \qquad (1.3)$$

**Input:**

```
.EQ (2.2a)
bold V bar sub nu~=~left [ pile {a above b above
c } right ] + left [ matrix { col { A(11) above .
above . } col { . above . above .} col {. above .
above A(33) }} right ] cdot left [ pile { alpha
above beta above  gamma } right ]
.EN
```

**Output:**

$$\bar{\mathbf{V}}_{\nu} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} A(11) & . & . \\ . & . & . \\ . & . & A(33) \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{2.2a}$$

**Input:**

```
.EQ I "" 2.75i
F hat ( chi ) ~ mark = ~ | del V | sup 2
.EN
.EQ I "" 2.75i
lineup =~ {left ( {partial V} over {partial x} right ) } sup 2
+ { left ( {partial V} over {partial y} right ) } sup 2
~~~~~~ lambda -> inf
.EN
```

**Output:**

$$\hat{F}(\chi) = |\nabla V|^2$$

$$= \left( \frac{\partial V}{\partial x} \right)^2 + \left( \frac{\partial V}{\partial y} \right)^2 \qquad \lambda \to \infty$$

**Input:**

```
$ a dot $,  $ b dotdot$,  $ rho tilde~times~y vec$.
```

**Output:**

$\dot{a}$, $\ddot{b}$, $\tilde{\rho} \times \vec{y}$.
(with delim $$ on).

## Registers You Can Change

Table 1-2 *Registers You Can Change*

| Register Name | Controls | Default Setting | Command to Change | Takes Effect Next |
|---|---|---|---|---|
| `LL` | Line length of text | 6 inches (6i) | `.nr LL 7.5i` | paragraph |
| `LT` | Length of titles | LL | `.nr LT 5i` | page |
| `FL` | Line length of footnotes | 5.5i | `.nr FL LL` | `.FS` request |
| `FI` | Footnote indent | 5 ens (5n) | `.nr FI 2n` | `.FS` request |
| `PS` | Point size | 10 | `.nr PS 11` | paragraph |
| `VS` | Vertical spacing | 12 | `.nr VS 13` | paragraph |
| `CW` | Column width | LL * 7/15 | `.nr CW 3i` | `.2C` or `.MC` request |
| `GW` | Intercolumn spacing | LL * 1/15 | `.nr GW .5i` | `.2C` or `.MC` request |
| `HM` | Header margin | 1i | `.nr HM .75i` | page |
| `FM` | Footer margin | 1i | `.nr FM .75i` | page |
| `PI` | Paragraph indent | 5 ens (5n) | `.nr PI 2n` | paragraph |
| `PD` | Paragraph depth | .3 vertical space (.3v) | `.nr PD 0` | paragraph |
| `QI` | Left and right indent for quote paragraph (`.QP`) | 5n | `.nr QI 8n` | `.QP` request |
| `DD` | Vertical distance around displays | .5v | `.nr DD 1v` | `.DS` request |
| `PO` | Page offset | 1i | `.nr PO 0.5i` | page |
| `LH` | Left page header | null | `.ds LH Sun` | page |
| `CH` | Center page header | null | `.ds CH Confidential` | page |
| `RH` | Right page header | null | `.ds RH Software` | page |
| `LF` | Left page footer | null | `.ds LF Do Not Copy` | page |
| `CF` | Center page footer | page number register (–\nPN–) | `.ds CF Draft` | page |
| `RF` | Right page footer | null | `.ds RF %` | page |
| `%` | Page number | 1 | `.nr % 3` | page |

# 2

# Formatting Documents with the -ms Macros

# 2

# Formatting Documents with the -ms Macros

This chapter describes the -ms macro package for preparing documents with `nroff` and `troff` on the Sun system.[1] The -ms Request Summary at the end of this chapter provides a quick reference for all the -ms macros and for useful displaying and printing commands. If you are acquainted with -ms, there is a quick reference for the *new* requests and string definitions as well. The differences between the new and the old -ms macro packages are described in the section entitled "Changes in the New -ms Macro Package." The section "Displaying and Printing Documents with -ms" describes how you can produce documents on either your workstation, printer, or phototypesetter without changing the text and formatting request input.

## 2.1. Changes in the New -ms Macro Package

The old -ms macro package has been revised, and the new macro package assumes the name -ms. There are some extensions to previous -ms macros and a number of new macros, but all the previously documented -ms macros still work exactly as they did before, and have the same names as before. The new -ms macro package includes several bug fixes, including a problem with the single-column `.1C` macro, minor difficulties with boxed text, a break induced by `.EQ` before initialization, the failure to set tab stops in displays, and several bothersome errors in the `refer` bibliographic citation macros. Macros used only at Bell Laboratories have been removed from the new version. We list them at the end of this chapter in the

## 2.2. Displaying and Printing Documents with -ms

After you have prepared your document with text and -ms formatting requests and stored it in a file, you can display it on your workstation screen or print it with `nroff` or `troff` with the -ms option to use the -ms macro package. A good way to start is to pipe your file through `more` for viewing:

```
hostname% nroff -ms filename ...   | more
```

If you forget the -ms option, you get continuous, justified, unpaginated output in which -ms requests are ignored. You can format more than one file on the

[1] The material in this chapter is derived from *A Revised Version of* -ms, B. Tuthill, University of California, Berkeley; *Typing Documents on the UNIX System: Using the* -ms *Macros with* `troff` *and* `nroff`, M.E. Lesk, Bell Laboratories, Murray Hill, New Jersey; and *Document Formatting on UNIX: Using the* -ms *Macros*, Joel Kies, University of California, Berkeley.

command line at a time, in which case `nroff` simply processes all of them in the order they appear, as if they were one file. There are other *options* to use with `nroff` and `troff`; see the *SunOS Reference Manual* for details.

You can get preview and final output of various sorts with the following commands. To send `nroff` output to the line printer, type:

```
hostname% nroff -ms filename | lpr -Pprinter
```

To produce a file with tables, use:

```
hostname% tbl filename | nroff -ms | lpr -printer
```

To produce a file with equations, type:

```
hostname% neqn filename | nroff -ms | lpr -printer
```

To produce a file with tables and equations, use the following order:

```
hostname% tbl filename | neqn | nroff -ms | lpr -printer
```

To print your document with `troff`, use:

```
hostname% troff -ms filename | lpr -t -printer
```

See `lpr(1)` in the *SunOS Reference Manual* for details on printing.

## 2.3. What Can Macros Do?

Macros can help you produce paragraphs, lists, sections (optionally with automatic numbering), page titles, footnotes, equations, tables, two-column format, a table of contents, endnotes, running heads and feet, and cover pages for papers. As with other formatting utilities such as `nroff` and `troff`, you prepare text interspersed with formatting requests. However, the macro package, which itself is written in `troff` commands, provides higher-level commands than those provided with the basic `troff` program. In other words, you can do a lot more with just one macro than with one `troff` request.

## 2.4. Formatting Requests

An `-ms` request usually consists of one or two upper-case characters, and usually in the form `.`*XX*.

The easiest way to produce simple formatted text is to put a `.LP` request on a line by itself at the beginning of the document. Add your text, on the following lines, leaving just a blank line to separate paragraphs. The `.LP` request produces a left-blocked paragraph, as we used throughout this chapter. Your output will have paragraphs and be paginated with right and left-justified margins.

When you use a macro package, you type in text as you normally do and intersperse it with formatting *requests*. For example, instead of spacing in with the space bar or typing a tab to indent paragraphs, put a `.PP` request on a line by itself before each paragraph. When formatted, this indents the first line of the following paragraph.

*Note:* You cannot just begin a document with a line of text. You must include an -ms request before any text input. When in doubt, use .LP to properly *initialize* the file, although any of the requests .PP, .LP, .TL, .SH, .NH is good enough. See the section "Cover Sheets and Title Pages" later in this chapter for the correct arrangement of requests at the start of a document.

## Paragraphs

You can produce several different kinds of paragraphs with the -ms macro package: standard, left-block, indented, labeled, and quoted.

### Standard Paragraph — .PP

To get an ordinary paragraph, use the .PP request, followed on subsequent lines by the text of the paragraph. For example, you type:

```
.PP
Tom appeared on the sidewalk with a bucket of whitewash and a long-handled
brush.
He surveyed the fence, and all gladness left him and a deep melancholy
settled down upon his spirit.
Thirty yards of board fence nine feet high.
Life to him seemed hollow, and
existence but a burden.
```

to produce:

Tom appeared on the sidewalk with a bucket of whitewash and a long-handled brush. He surveyed the fence, and all gladness left him and a deep melancholy settled down upon his spirit. Thirty yards of board fence nine feet high. Life to him seemed hollow, and existence but a burden.

### Left-Block Paragraph — .LP

You can also produce a left-block paragraph, like those in this manual, with .LP. The first line is not indented as it is with the .PP request. For example, you type:

```
.LP
Tom appeared ...
```

to produce:

Tom appeared on the sidewalk with a bucket of whitewash and a long-handled brush. He surveyed the fence, and all gladness left him and a deep melancholy settled down upon his spirit. Thirty yards of board fence nine feet high. Life to him seemed hollow, and existence but a burden.

There are default values for the vertical spacing before paragraphs and for the width of the indentation. To change the paragraph spacing, see the section "Modifying Default Features."

### Indented Paragraph — .IP

Another kind of paragraph is the indented paragraph, produced by the .IP request. These paragraphs can have hanging numbers or labels. For example:

```
.IP [1]
Text for first paragraph, typed
normally for as long as you would
like on as many lines as needed.
.IP [2]
Text for second paragraph, ...
.LP
```

produces

[1] Text for first paragraph, typed normally for as long as you would like on as many lines as needed.

[2] Text for second paragraph, ...

A series of indented paragraphs must be followed by an ordinary paragraph beginning with .PP or .LP, depending on whether you wish indenting or not. Here we used the .LP request.

More sophisticated uses of .IP are also possible. If the label is omitted, for example, you get a plain block indent:

```
Tom appeared on the sidewalk with a bucket of whitewash and a long-handled
brush.
.IP
He surveyed the fence, and all gladness left him and a deep melancholy
settled down upon his spirit.
Thirty yards of board fence nine feet high.
Life to him seemed hollow, and
existence but a burden.
.LP
```

which produces

Tom appeared on the sidewalk with a bucket of whitewash and a long-handled brush.

> He surveyed the fence, and all gladness left him and a deep melancholy settled down upon his spirit. Thirty yards of board fence nine feet high. Life to him seemed hollow, and existence but a burden.

If a non-standard amount of indenting is required, specify it after the label in character positions. It remains in effect until the next .PP or .LP. Thus, the general form of the .IP request contains two additional fields: the label and the indenting length. For example,

```
.IP "Example one:" 15
Notice the longer label, requiring larger
indenting for these paragraphs.
.IP "Example two:"
And so forth.
.LP
```

produces this:

Example one:     Notice the longer label, requiring larger indenting for these paragraphs.

Example two:     And so forth.

Notice that you must enclose the label in double quote marks because it contains a space; otherwise, the space signifies the end of the argument. The indentation request above is in the number of *ens*, a unit of dimension used in typesetting. An *en* is approximately the width of a lowercase 'n' in the particular point size you are using.

The .IP macro adjusts properly by causing a break to the next line if you type in a label longer than the space you allowed for. For example, if you have a very long label and have allowed 10 en-spaces for it, your input looks like:

```
.IP "A very, very, long and verbose label" 10
And now here's the text that you want.
And now here's the text that you want.
And now here's the text that you want.
And now here's the text that you want.
And now here's the text that you want.
```

And your output is adjusted accordingly with a break between the label and the text body:

A very, very, long and verbose label

> And now here's the text that you want. And now here's the text that you want. And now here's the text that you want. And now here's the text that you want. And now here's the text that you want.

### Nested Indentation — .RS and .RE

It is also possible to produce multiple (or *relative*) nested indents; the .RS request indicates that the next .IP starts its indentation from the current indentation level. Each .RE undoes one level of indenting, so you should balance .RS and .RE requests. Think of the .RS request as 'move right' and the .RE request as 'move left'. As an example:

```
.IP I.
South Bay Area Restaurants
.RS
.IP A.
Palo Alto
.RS
.IP 1.
La Terrasse
.RE
.IP B.
Mountain View
.RS
.IP 1.
Grand China
.RE
.IP C.
Menlo Park
.RS
.IP 1.
Late for the Train
.IP 2.
Flea Street Cafe
.RE
.RE
.LP
```

results in:

I. South Bay Area Restaurants

 A. Palo Alto

  1. La Terrasse

 B. Mountain View

  1. Grand China

 C. Menlo Park

  1. Late for the Train

  2. Flea Street Cafe

Note the two `.RE` requests in a row at the end of the list. Remember that you need one *end* for each *start*.

### Quoted Paragraph — `.QP`

All of the variations on `.LP` leave the right margin untouched. Sometimes, you need a a paragraph indented on both right and left sides. To set off a quotation as such, use:

```
.QP
Precede each paragraph that you want offset as a quotation
with a .QP. This produces a paragraph like this.
Notice that the right edge is also indented from the right margin.
```

to produce

> Precede each paragraph that you want offset as a quotation with a .QP. This produces a paragraph like this. Notice that the right edge is also indented from the right margin.

### Section Headings — .SH and .NH

There are two varieties of section headings, unnumbered with .SH and numbered with .NH. In either case, type the text of the section heading on one or more lines following the request. End the section heading by typing a subsequent paragraph request or another section heading request. When printed, one line of vertical space precedes the heading, which begins at the left margin. `nroff` offsets the heading with blank lines, while `troff` sets it in **boldface** type. .NH section headings are numbered automatically. The macro takes an argument number representing the *level-number* of the heading, up to 5. A third-level section number is one like '1.2.1'. The macro adds one to the section number at the requested level, as shown in the following example:

```
.NH
Bay Area Recreation
.NH 2
Beaches
.NH 3
San Gregorio
.NH 3
Half Moon Bay
.NH 2
Parks
.NH 3
Wunderlich
.NH 3
Los Trancos
.NH 2
Amusement Parks
.NH 3
Marine World/Africa USA
```

generates:

2. Bay Area Recreation

2.1 Beaches

2.1.1 San Gregorio

2.1.2 Half Moon Bay

2.2 Parks

2.2.1 Wunderlich

2.2.2 Los Trancos

2.3 Amusement Parks

2.3.1 Marine World/Africa USA

`.NH` without a level-number means the same thing as `.NH 1`, and `.NH 0` cancels the numbering sequence in effect and produces a section heading numbered 1.

### Cover Sheets and Title Pages — `.TL` and `.AU`

`-ms` provides a group of macros to format items that typically appear on the cover sheet or title page of a formally laid-out paper. You can use them selectively, but if you use several, you must put them in the order shown below, normally at or near the beginning of the input file.

The first line of a document signals the general format of the first page. In particular, if it is `.RP` (released paper), a cover sheet with title and abstract is prepared. The default format is useful for scanning drafts.

Sample input is:

```
.RP         (Optional; use for released paper format)
.TL
Title of document (one or more lines)
.AU
Author(s)          (may also be several lines)
.AI
Author's institution(s)
.AB
Abstract; to be placed on the cover sheet of a paper.
Line length is 5/6 of normal; use .ll here to change.
.AE         (abstract end)
text ...           (begins with .PP)
```

(See *Order of Requests in Input* for a quick example of this scheme.)

If the .RP request precedes .TL, the title, author, and abstract material are printed separately on a cover sheet. The title and author information (not the abstract) is then repeated automatically on page one (the title page) of the paper, without your having to type it again. If you do not include an .RP request, all of this material appears on page one, followed on the same page by the main text of the paper.

To omit some of the standard headings (such as no abstract, or no author's institution), just omit the corresponding fields and command lines. To suppress the word ABSTRACT type .AB no rather than .AB. You can intersperse several .AU and .AI lines to format for multiple authors.

These macros are optional; you may begin a paper simply with a section heading or paragraph request. When you do precede the main text with cover sheet and title page material, include a paragraph or section heading between the last title page request and the beginning of the main text. Don't forget that some -ms request must precede any input text.

### Running Heads and Feet — LH, CH, RH

The -ms macros, by default, print a page heading containing a page number (if greater than 1). You can make minor adjustments to the page headings and footings by redefining the strings LH, CH, and RH which are the left, center and right portions of the page headings, respectively; and the strings LF, CF, and RF, which are the left, center and right portions of the page footer. For nroff output, there are two default values: CH is the current page number surrounded on either side by hyphens, and CF contains the current date as supplied by the computer. For troff CH also contains the page number, but CF is empty. The other four registers are empty by default for both nroff and troff. You can use the .ds request to assign a value to a string register. For example:

```
.ds RF Draft Only \(em Do Not Distribute
```

This prints the character string

Draft Only — Do Not Distribute

at the bottom right of every page. You do not need to enclose the string in double quote marks. To remove the contents of a string register, simply redefine it as empty. For instance, to clear string register CH, and make the center header blank on the following pages, use the request:

```
.ds CH
```

To put the page number in the right header, use:

```
.ds RH %
```

In a string definition, '%' is a special symbol referring to `nroff`'s automatic page counter. If you want hyphens on either side of the page number, place them on either side of the '%' in the command, that is:

```
.ds RH -%-
```

Remember that putting the page number in the right header as shown above does not remove it from the default CH; you still have to clear out CH.

If you want requests that set the values of string and number registers to take effect on the first page of output, put them at or near the beginning of the input file, before the initializing macro, which in turn must precede the first line of text. Among other functions, the initializing macro causes a 'pseudo page break' onto page one of the paper, including the top-of-page processing for that page. Be sure to put requests that change the value of the PO (page offset), HM (top or head margin), and FM (bottom or foot margin) number registers and the page header string registers before the transition onto the page where they are to take effect.

For more complex formats, you can redefine the macros PT (page top) and BT (page bottom), which are invoked respectively at the top and bottom of each page. The margins (taken from registers HM and FM for the top and bottom margin respectively) are normally 1 inch; the page header/footer are in the middle of that space. If you redefine these macros, be careful not to change parameters such as point size or font without resetting them to default values.

### Custom Headers and Footers — `.OH`, `.EH`, `.OF`, and `.EF`

You can also produce custom headers and footers that are different on even and odd pages. The `.OH` and `.EH` macros define odd and even headers, while `.OF` and `.EF` define odd and even footers. Arguments to these four macros are specified as with the `nroff .tl`, that is, there are three fields (left, center and right), each separated by a single apostrophe. For example, to get odd-page headers with the chapter name followed by the page number and the reverse on even pages, use:

```
.OH   ´ For Whom the Bell Tolls´ ´ Page  %´
.EH   ´ Page  %´ ´ For Whom the Bell Tolls´
```

Note that it is an error to have an apostrophe in the header text; if you need an apostrophe, use a backslash and apostrophe (´) or a delimiter other than apostrophe around the left, center, and right portions of the title. You can use any character as a delimiter, provided it doesn't appear elsewhere in the argument to .OH, .EH, .OF, or .EF.

You can use the .P1 (dot-P-one) macro to print the header on page 1. If you want roman numeral page numbering, use an .af PN i request.

### Multi-Column Formats — .2C and .MC

If you place the request .2C in your document, the document will be printed in double column format beginning at that point. This is often desirable on the typesetter. Each column will have a width 7/15 that of the text line length in single-column format, and a gutter (the space between the columns) of 1/15 of the full line length. Remember that when you use the two-column .2C request, either pipe the nroff output through col or make the first line of the input .pi /usr/bin/col.

The .2C request is actually a special case of the .MC request that produces formats of more than two spaces:

```
.MC  [column width  [ gutter width  ]  ]
```

This formats output in as many columns of *column width* as will fit across the page with a gap of *gutter width*. You can specify the column width in any unit of scale, but if you do not specify a unit, the setting defaults to ens. .MC without any column width is the same thing as .2C. For example:

```
.MC
Tom appeared on the sidewalk with a bucket of whitewash and a long-handled
brush.
He surveyed the fence, and all gladness left him and a deep melancholy
settled down upon his spirit.
```

To return to single-column output, use .1C. Switching from double to single-column always causes a skip to a new page.

## Footnotes — .FS and .FE

Material placed between lines with the commands .FS (footnote start) and .FE (footnote end) is collected, remembered, and placed at the bottom of the current page.* The formatting of the footnote is:

```
...at the bottom of the current page.*
.FS
* Like this.
.FE
```

By default, footnotes are 11/12th the length of normal text, but you can modify this by changing the FL register (see the "Modifying Default Features" section). When typeset, footnotes appear in smaller size type.

Because the macros only save a passage of text for printing at the bottom of the page, you have to mark the footnote reference in some way, both in the text preceding the footnote and again as part of the footnote text. We use a simple asterisk, but you can use anything you want.

You can also produce automatically-numbered footnotes. Footnote numbers are printed by a predefined string (**), which you invoke separately from .FS and .FE. Each time this string is used, it increases the footnote number by one, whether or not you use .FS and .FE in your text. Footnote numbers are superscripted on the phototypesetter and on daisy-wheel terminals, but on low-resolution devices (such as the line printer and a treminal), they are bracketed. If you use ** to indicate numbered footnotes, the .FS macro automatically includes the footnote number at the bottom of the page. This footnote, for example, was produced as follows:[2]

```
This footnote, for example, was produced as follows:\**
.FS
If you never use the ...
.FE
```

If you are using ** to number footnotes, but want a footnote of the same style marked with an asterisk or dagger, give that mark as the first argument to .FS:†

```
give that mark as the first argument to .FS:\(dg
.FS \(dg
In the footnote, the dagger ...
.FE
```

Footnote numbering is temporarily suspended, because the ** string is not used. Instead of a dagger, you could use an asterisk * or double dagger ‡, represented as \(dd.

* Like this.

[2] If you never use the ** string, no footnote numbers will appear anywhere in the text, including down here. The output footnotes will look exactly like footnotes produced with **-mos**, the old -ms macro package.

† In the footnote, the dagger will appear where the footnote number would otherwise appear, as shown here.

### Endnotes

If you want to produce endnotes rather than footnotes, put the references in a file of their own. This is similar to what you would do if you were typing the paper on a conventional typewriter. Note that you can use automatic footnote numbering without actually having the `.FS` and `.FE` pairs in your text. If you place footnotes in a separate file, you can use `.IP` macros with ** as a hanging tag; this gives you numbers at the left-hand margin. With some styles of endnotes, you would want to use `.PP` rather than `.IP` macros, and specify ** before the reference begins.

### Displays and Tables — .DS and .DE

To prepare displays of lines, such as tables, in which the lines should not be rearranged or broken between pages, enclose them in the requests `.DS` and `.DE`:

```
.DS
lines, like the
examples here, are placed
between .DS and .DE macros
.DE
```

which produces:

> lines, like the
> examples here, are placed
> between .DS and .DE macros

By default, lines between `.DS` and `.DE` are indented from the left margin.

If you don't want the indentation, use `.DS L` to begin and `.DE` to produce a left-justified display:

to get
something like
this

You can also center lines with the `.DS C` and `.DE` requests:

This is an
example
of a centered display.

Note that each line is centered individually.

A plain `.DS` is equivalent to `.DS I`, which indents and left-adjusts. An extra argument to the `.DS I` or `.DS` request is taken as an amount to indent. For example, `.DS I 3` or `.DS 3` begins a display to be indented 3 ens from the margin.

There is a variant `.DS B` that makes the display into a left-adjusted block of text, and then centers that entire block.

Normally a display is kept together on one page. If you wish to have a long display which may be split across page boundaries, use `.CD`, `.LD`, and `.BD` in place of the requests `.DS C`, `.DS L`, and `.DS B` respectively. Use `.ID` for either a plain `.DS` or `.DS I`. You can also specify the amount of indentation with the `.ID` macro. Use the following table as a quick reference:

Table 2-1 *Display Macros*

| Macro with Keep | Macro without Keep |
|---|---|
| `.DS I` | `.ID` |
| `.DS L` | `.LD` |
| `.DS C` | `.CD` |
| `.DS B` | `.BD` |
| `.DS` | `.ID` |

*Note:* It is tempting to assume that `.DS R` will right-adjust lines, but it doesn't.

### Keeping Text Together — `.KS`, `.KE` and `.KF`

If you wish to keep a table or other block of lines together on a page, there are 'keep - release' requests. If a block of lines preceded by `.KS` and followed by `.KE` does not fit on the remainder of the current page, it will begin on a new page. There is also a 'keep floating' request. If the block to be kept together is preceded by `.KF` instead of `.KS` and does not fit on the current page, it will be moved down through the text to the top of the next page. `nroff` fills in the current page with the ordinary text that follows the keep in the input file to avoid leaving blank space at the bottom of the page preceding the keep. Thus, no large blank space will be introduced in the document.

In multi-column output, the keep macros attempt to place all the kept material in the same column. If the material enclosed in a keep requires more than one page, or more than a column in multi-column format, it will start on a new page or column and simply run over onto the following page or column.

### Boxing Words or Lines — `.BX` and `.B1` and `.B2`

To draw rectangular boxes around words, use the request

```
.BX word
```

to print $\boxed{\text{word}}$ as shown. You can box longer pieces of text by enclosing them with `.B1` and `.B2`:

```
.B1
Tom appeared on the sidewalk with a bucket of whitewash
and a long-handled brush.
He surveyed the fence, and all gladness left him and a deep melancholy
settled down upon his spirit.
Thirty yards of board fence nine feet high.
Life to him seemed hollow, and
existence but a burden.
.B2
```

This produces:

```
Tom appeared on the sidewalk with a bucket of whitewash and a long-handled
brush. He surveyed the fence, and all gladness left him and a deep melancholy
settled down upon his spirit. Thirty yards of board fence nine feet high. Life to
him seemed hollow, and existence but a burden.
```

### Changing Fonts — .I, .B, .R and .UL

To get italics on the typesetter or reverse display on the workstation, say:

```
.I
as much text as you want
can be typed here
.R
```

as was done for *these three words*. The .R request restores the normal (usually Roman) font. If only one word is to be italicized, you can put it on the line with the .I request:

```
.I word
```

and in this case you do not need to use an .R to restore the previous font.

You can print **boldface** font by

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

As with .I, you can place a single word in boldface font by putting it on the same line as the .B request. Also, when .I or .B is used with a word as an argument, it can take as a second argument any trailing punctuation to be printed immediately after the word but set in normal typeface. For example:

```
.B word )
```

prints

**word**)

that is, the word in boldface and the closing parenthesis in normal Roman directly adjacent to the word.

If you want actual underlining as opposed to italicizing on the typesetter, use the request

```
.UL word
```

to underline a word. There is no way to underline multiple words on the typesetter.

### Changing the Type Size — .LG, .SM and .NL

You can specify a few size changes in troff output with the requests .LG (make larger), .SM (make smaller), and .NL (return to normal size). The size change is two points (see the "Dimensions" section for a discussion of point size); you can repeat the requests for increased effect (here one .NL canceled two .SM requests). These requests are primarily useful for temporary size changes for a small number of words. They do not affect vertical spacing of lines of text. See the section on "Modifying Default Features" for other techniques for changing the type size and vertical spacing of longer passages.

### Dates — `.DA` and `.ND`

When you use `-ms`, `nroff` prints the date at the bottom of each page, but `troff` does not. Both `nroff` and `troff` print it on the cover sheet if you have requested one with `.RP`. To make `troff` print the date as the center page footer, say `.DA` (date). To suppress the date, say `.ND` (no date). To lie about the date, type `.DA July 4, 1776`, which puts the specified date at the bottom of each page. The request:

```
.ND September 16, 1959
```

in `.RP` format places the specified date on the cover sheet and nowhere else. Place either `.ND` or `.DA` before the `.RP` Notice this is one instance that you do not need to put double quote marks around the arguments.

### Thesis Format Mode — `.TM`

To format a paper as a thesis, use the `.TM` macro (thesis mode). It is much like the `.th` macro in the **-me** macro package. It puts page numbers in the upper right-hand corner, numbers the first page, suppresses the date, and doublespaces everything except quotes, displays, and keeps. Use it at the top of each file making up your thesis. Calling `.TM` defines the `.CT` macro for chapter titles, which skips to a new page and moves the page number to the center footer. You can use the `.P1` (P one) macro even without thesis mode to print the header on page one, which is suppressed except in thesis mode. If you want roman numeral page numbering, use an `.af PN i` request.

### Bibliography — `.XP`

To format bibliography entries, use the `.XP` macro, which stands for *exdented paragraph*. It exdents the first line of the paragraph by \n(PI units, usually 5n, the same as the indent for the first line of a `.PP`. An example of exdented paragraphs is:

```
.XP
Lumley, Lyle S., \fISex in Crustaceans: Shell Fish Habits,\fP\^
Harbinger Press, Tampa Bay and San Diego, October 1979.
243 pages.
The pioneering work in this field.
.XP
Leffadinger, Harry A., "Mollusk Mating Season: 52 Weeks, or All Year?"
in \fIActa Biologica,\fP\^ vol. 42, no. 11, November 1980.
A provocative thesis, but the conclusions are wrong.
```

which produces:

Lumley, Lyle S., *Sex in Crustaceans: Shell Fish Habits,* Harbinger Press, Tampa Bay and San Diego, October 1979. 243 pages. The pioneering work in this field.

Leffadinger, Harry A., "Mollusk Mating Season: 52 Weeks, or All Year?" in *Acta Biologica,* vol. 42, no. 11, November 1980. A provocative thesis, but the conclusions are wrong.

You do have to italicize the book and journal titles and quote the title of the journal article. You can change the indentation and exdentation by setting the value of number register PI.

**Table of Contents — .XS, .XE, .XA, .PX**

There are four macros that produce a table of contents. Enclose table of contents entries in .XS and .XE pairs, with optional .XA macros for additional entries. Arguments to .XS and .XA specify the page number, to be printed at the right. A final .PX macro prints out the table of contents. A sample of typical input and output text is:

```
.XS  ii
Introduction
.XA  1
Chapter 1: Review of the Literature
.XA  23
Chapter 2: Experimental Evidence
.XE
.PX
```

**Table of Contents**

| | |
|---|---|
| Introduction ........................................................................................ | ii |
| Chapter 1: Review of the Literature ........................................................ | 1 |
| Chapter 2: Experimental Evidence .......................................................... | 23 |

You can also use the .XS and .XE pairs in the text, after a section header for instance, in which case page numbers are supplied automatically. However, most documents that require a table of contents are too long to produce in one run, which is necessary if this method is to work. It is recommended that you make the table of contents after finishing your document. To print out the table of contents, use the .PX macro or nothing will happen.

**Defining Quotation Marks**

To produce quotation marks and dashes that format correctly with both `nroff` and `troff`, there are some string definitions for each of the formatting programs. The *— string yields two hyphens in `nroff`, and produces an em-dash — like this one in `troff`. The *Q and *U strings produce “ and ” in `troff`, but " in `nroff`.

**Accent Marks**

To simplify typing certain foreign words, the -ms macro package defines strings representing common accent marks. There are a large number of optional foreign accent marks defined by the -ms macros. All the accent marks available in **-mos** are present, and they all work just as they always did.

For the old accent marks, type the string *before* the letter over which the mark is to appear. For example, to print ‘t´el´ephone with the old macros, you type:

```
t\*'el\*'ephone
```

Unlike the old accent marks, the new accent strings should be placed *after* the letter being accented. Place `.AM` (accent mark) at the beginning of your document, and type the accent strings *after* the letter being accented. A list of both sets of diacritical marks and examples of what they look like follows. *Note:* Do not use the `tbl` macros `.TS` and `.TE` with any of the accent marks as the marks do not line up correctly.

Table 2-2 *Old Accent Marks*

| Accent Name | Input | Output |
|---|---|---|
| acute | `\*´e` | é |
| grave | `` \*`e `` | è |
| umlaut | `\*:u` | ü |
| circumflex | `\*^e` | ê |
| tilde | `\*~a` | ã |
| haček | `\*Cr` | ř |
| cedilla | `\*,c` | c |

Table 2-3 *Accent Marks*

| Accent Name | Input | Output |
|---|---|---|
| acute | `e\*´` | é |
| grave | `` e\*` `` | è |
| circumflex | `o\*^` | ô |
| cedilla | `c\*,` | ç |
| tilde | `n\*~` | ñ |
| question | `\*?` | ¿ |
| exclamation | `\*!` | ¡ |
| umlaut | `u\*:` | ü |
| digraphes | `\*8` | β |
| haček | `c\*v` | č |
| macron | `a\*_` | ā |
| o-slash | `o\*/` | ø |
| yogh | `kni\*3t` | kniȝt |
| angstrom | `a\*o` | å |
| Thorn | `\*(Th` | Þ |
| thorn | `\*(th` | þ |
| Eth | `\*(D-` | Đ |
| eth | `\*(d-` | ∂ |
| hooked o | `\*q` | ǫ |
| ae ligature | `\*(ae` | æ |
| AE ligature | `\*(Ae` | Æ |
| oe ligature | `\*(oe` | œ |
| OE ligature | `\*(Oe` | Œ |

If you want to use these new diacritical marks, don't forget the `.AM` at the top of your file. Without it, some of these marks will not print at all, and others will be placed on the wrong letter.

## 2.5. Modifying Default Features

The -ms macro package supplies a standard page layout style. The text line has a default length of six inches; the indentation of the first line of a paragraph is five ens; the page number is printed at the top center of every page after page one; and so on for standard papers. You can alter many of these default features by changing the values that control them.

The computer memory locations where these values are stored are called *number registers* and *string registers*. Number and string registers have names like those of requests, one or two characters long. For instance, the value of the line length is stored in a number register named LL. Unless you give a request to change the value stored in register LL, it will contain the standard or default value assigned to it by -ms. The "Summary of -ms Number Registers" table lists the number registers you can change along with their default values.

### Dimensions

To change a dimension like the line length from its default value, reset the associated number register with the troff request .nr (number register):

```
.nr LL 5i
```

The first argument, LL, is the name of a number register, and the second, 5i is the value being assigned to it. In the case above, the line length is adjusted from the default six inches to five inches. As another example, consider:

```
.nr PS 9
```

which makes the default point size 9 point.

The value may be expressed as an integer or may contain a decimal fraction. When setting the value of a number register, it is almost always necessary to include a unit of scale immediately after the value. In the example above, the 'i' as the unit of scale lets troff know you mean five inches and not five of some other unit of distance. But the point size (PS) and vertical spacing (VS) registers are exceptions to this rule; ordinarily they should be assigned a value as a number of points *without indicating the unit of scale.* For example, to set the vertical spacing to 24 points, or one-third of an inch (double-spacing), use the request:

```
.nr VS 24
```

In the unusual case where you want to set the vertical spacing to more than half an inch (more than 36 points), include a unit of scale in setting the VS register. The "Units of Measurement in nroff and troff" table explains the units of measurement.

Table 2-4 *Units of Measurement in* `nroff` *and* `troff`

| ***Unit*** | ***Abbr*** | `nroff` | `troff` |
|---|---|---|---|
| point | p | 1/72 inch | 1/72 inch |
| pica | p | 1/6 inch | 1/6 inch |
| em | m | width of one character | distance equal to number of points in the current typesize |
| en | n | width of one character | half an em |
| vertical space | v | amount of space in which each line of text is set, measured baseline to baseline | same |
| inch | i | inch | inch |
| centimeter | c | centimeter | centimeter |
| machine unit | u | 1/240 inch | 1/432 inch |

The units *point*, *pica*, *em*, and *en* are units of measurement used by tradition in typesetting. The *vertical space* unit also corresponds to the typesetting term *leading*, which refers to the distance from the baseline of one line of type to the baseline of the next. Em and en are particularly interesting in that they are proportional to the type size currently in use (normally expressed as a number of points). An em is the distance equal to the number of points in the type size (roughly the width of the letter 'm' in that point size), while an en is half that (about the width of the letter 'n'). These units are convenient for specifying dimensions such as indentation. In `troff`, em and en have their traditional meanings, that is one em of distance is equal to two ens. For `nroff`, on the other hand, em and en both mean the same quantity of distance, the width of one typewritten character.

The *machine unit* is a special unit of dimension used by `nroff` and `troff` internally. This is the unit to which the programs convert almost all dimensions when storing them in memory, and is included here primarily for completeness. In using the features of `-ms`, it is sufficient to know that such a unit of measure exists.

Note that a change to a number register such as LL does not immediately change the related dimension at that point in the output. Instead, in the case of the line length for example, the change takes place at the beginning of the next paragraph, where `-ms` resets various dimensions to the current values of the related number registers.

If you need the effect immediately, use the normal `troff` command in addition to changing the number register. For example, to control the vertical spacing immediately, use:

```
.vs
```

This takes effect at the place where it occurs in your input file. Since it does not change the VS register, however, its effect lasts only until the beginning of the next paragraph. As a general rule, to make a permanent change, or one that will last for several paragraphs until you want to change it again, alter the value of the -ms register. If the change must happen immediately, somewhere other than the point shown in the table, use the `troff` request. If you want the change to be both immediate and lasting, do both.

Table 2-5 *Summary of* `-ms` *Number Registers*

| Register | Controls | Takes Effect | Default |
|---|---|---|---|
| PS | point size | next para. | 10 |
| VS | line spacing | next para. | 12 pts |
| LL | line length | next para. | 6″ |
| LT | title length | next para. | 6″ |
| PD | para. spacing | next para. | 0.3 VS |
| PI | para. indent | next para. | 5 ens |
| FL | footnote length | next `.FS` | 11/12 LL |
| CW | column width | next `.2C` | 7/15 LL |
| GW | intercolumn gap | next `.2C` | 1/15 LL |
| PO | page offset | next page | 26/27″ |
| HM | top margin | next page | 1″ |
| FM | bottom margin | next page | 1″ |

You may also alter the strings and which are the left, center, and right headings respectively; and similarly and which are strings in the page footer. Use the `troff .ds` (define string) request to alter the string registers, as you use the `.nr` request for number registers. The page number on *output* is taken from register to permit changing its output style. For more complicated headers and footers, you can redefine the macros and as explained earlier. See the "Register Names" section for a full list.

## 2.6. Using `nroff` and `troff` Requests

You can use a small subset of the `troff` requests to supplement the `-ms` macro package.

Use `.nr` and `.ds` requests to manipulate the `-ms` number and string registers as described in the "Modifying Default Features" section. You can also freely use the other following requests in a file for processing with the `-ms` macro package. They all work with both typesetter and workstation or terminal output.

`.ad b` Adjust both margins. This is the default adjust mode.

`.bp` Begin new page.

`.br` 'Break' line; start a new output line whether or not the current one has been completely filled with text.

`.ce` *n* Center the following *n* input text lines individually in the output. If *n* is omitted, only the next (one) line of text is centered.

`.ds` *XX* Define string register named *XX*.

`.na` Turn off adjusting of right margins to produce ragged right.

`.nr` *XX* Define number register named `XX`.

`.sp` ***n*** Insert *n* blank lines. If *n* is omitted, one blank line is produced (the current value of the unit *v*). You can attach a unit of dimension to *n* to specify the quantity in units other than a number of blank lines.

*Note:* The macro package executes sequences of `troff` requests on its own, in a manner invisible to you. By inserting your own `troff` requests, you run the risk of introducing errors. The most likely result is simply for your `troff` requests to be ignored, but in some cases the results can include fatal `troff` errors and garbled typesetter output.

As a simple example, if you try to produce a centered heading with the input:

```
.ce
.SH
Text of section heading
```

you will discover that the heading comes out left-adjusted; the `.SH` macro, appearing after the `.ce` request overrules it and forces left-adjusting. But consider the following sequence:

```
.sp
.ce
.B
Line of text
```

which successfully produces a centered, boldface heading preceded by one line of vertical space. There are lots of tricks like this, so be careful.

To learn more about `troff` see the chapter on "Formatting Documents with `nroff` and `troff`."

## 2.7. Using `-ms` with `eqn` to Typeset Mathematics

If you have to print Greek letters or mathematical equations, see the chapter "Typesetting Mathematics with `eqn`" for equation setting. To aid `eqn` users, `-ms` provides definitions of `.EQ` and `.EN` which normally center the equation and set it off slightly. An argument to `.EQ` is taken to be an equation number and placed in the right margin near the equation. In addition, there are three special arguments to `.EQ`: the letters `C`, `I`, and `L` indicate centered (default), indented, and left adjusted equations, respectively. If there is both a format argument and an equation number, give the format argument first, as in

```
.EQ L (1.3a)
```

for a left-adjusted equation numbered (1.3a).

## 2.8. Using `-ms` with `tbl` to Format Tables

Similar to the `eqn` macros are the macros `.TS` and `.TE` defined to separate tables from text with a little space (see the chapter "Formatting Tables with `tbl`"). A very long table with a heading may be broken across pages by beginning it with `.TS H` instead of `.TS`, and placing the line `.TH` in the table data after the heading. If the table has no heading repeated from page to page, just use the ordinary `.TS` and `.TE` macros.

## 2.9. Register Names

The `-ms` macro package uses the following register names internally. Independent use of these names in your own macros may produce incorrect output. Note that there are no lower case letters in any `-ms` internal name.

| Number Registers Used in -ms | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| : | DW | GW | HM | IQ | LL | NA | OJ | PO | T. | TV |
| #T | EF | H1 | HT | IR | LT | NC | PD | PQ | TB | VS |
| T. | FC | H2 | IF | IT | MF | ND | PE | PS | TC | WF |
| 1T | FL | H3 | IK | KI | MM | NF | PF | PX | TD | YE |
| AV | FM | H4 | IM | L1 | MN | NS | PI | RO | TN | YY |
| CW | FP | H5 | IP | LE | MO | OI | PN | ST | TQ | ZN |

| String Registers Used in -ms | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ´ | A5 | CB | DW | EZ | I | KF | MR | R1 | RT | TL |
| ` | AB | CC | DY | FA | I1 | KQ | ND | R2 | S0 | TM |
| ^ | AE | CD | E1 | FE | I2 | KS | NH | R3 | S1 | TQ |
| ~ | AI | CF | E2 | FJ | I3 | LB | NL | R4 | S2 | TS |
| : | AU | CH | E3 | FK | I4 | LD | NP | R5 | SG | TT |
| , | B | CM | E4 | FN | I5 | LG | OD | RC | SH | UL |
| 1C | BG | CS | E5 | FO | ID | LP | OK | RE | SM | WB |
| 2C | BT | CT | EE | FQ | IE | ME | PP | RF | SN | WH |
| A1 | C | D | EL | FS | IM | MF | PT | RH | SY | WT |
| A2 | C1 | DA | EM | FV | IP | MH | PY | RP | TA | XD |
| A3 | C2 | DE | EN | FY | IZ | MN | QF | RQ | TE | XF |
| A4 | CA | DS | EQ | HO | KE | MO | R | RS | TH | XK |

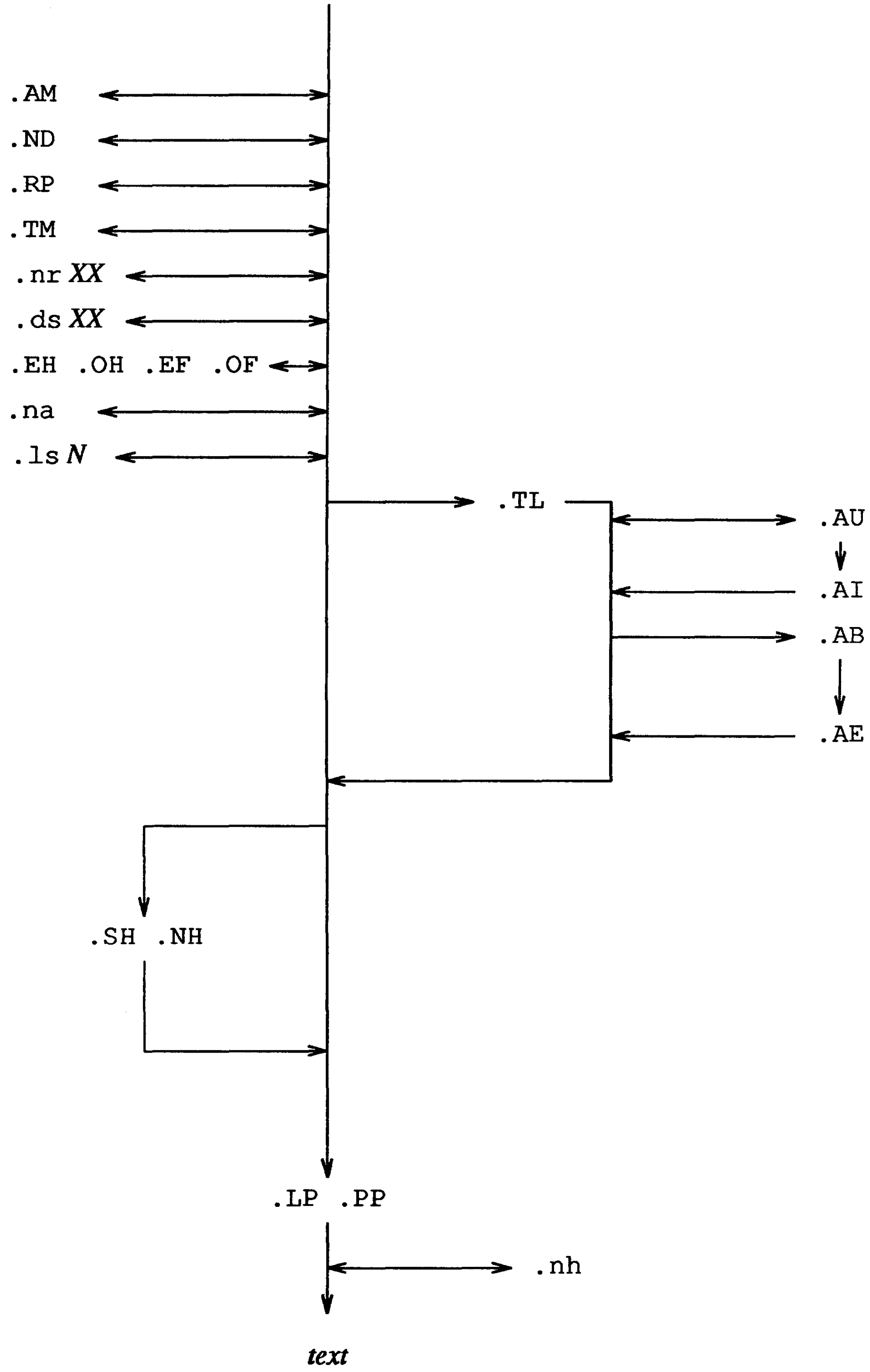
## 2.10. Order of Requests in Input

The following diagram provides a quick reference for how to order macro requests when using the `-ms` macro package to format documents. The central arrow indicates that the minimum formatting requests you need with `-ms` are the paragraph macros. These initialize certain quantities and are necessary to obtain predictable results when you use other macros.

The double-edged arrows indicate optional requests. The single-edged arrows indicate dependencies. For example, if you use a `.AB` request, you need a `.AE` request. If you use a `.AU` request, you don't need a `.AI` request, but if you use a `.AI` request, you have to use a `.AU` request first. The locations of the side arrows relative to the other requests indicate the relative locations of the requests in the document source.

For simpler documents, use just a .LP initializing request and just leave blank lines between paragraphs.

Figure 2-1 *Order of Requests in* -ms *Documents*



## 2.11. -ms Request Summary

This section includes tables of the old Bell Laboratories that have been removed from the new -ms package, of new -ms requests and string definitions, and of useful printing and displaying commands. It also includes a complete -ms request and string summary for easy reference.

Table 2-6 *Bell Laboratories Macros Deleted From* -ms

| Macro Request | Explanation |
|---|---|
| .CS | Cover sheet |
| .EG | BTL Engineer's Notes |
| .HO | Bell Labs, Holmdel, N.J. |
| .IH | Bell Labs, Naperville, Ill. |
| .IM | BTL internal memo |
| .MF | BTL file memo |
| .MH | Bell Labs, Murray Hill, N.J. |
| .MR | BTL record memo |
| .ND | BTL date |
| .OK | BTL keywords for tech memo |
| .PY | Bell Labs, Piscataway, N.J. |
| .SG | Signatures for tech memo |
| .TM | BTL technical memo |
| .TR | BTL report format |
| .WH | Bell Labs, Whippany, N.J. |

Table 2-7 *New* -ms *Requests*

| Macro Request | Explanation |
|---|---|
| .AM | New accent mark definitions. |
| .CT | Chapter title in .TM format. |
| .EH | Define even three-part page header. |
| .EF | Define even three-part page footer. |
| .FE | End automatically numbered footnote. |
| .FS | Begin automatically numbered footnote. |
| .IP ** | Endnotes with automatic numbering. |
| .IX | Index words. |
| .OF | Define odd three-part page footer. |
| .OH | Define odd three-part page header. |
| .P1 | Put header on page one in .TM format. |
| .PX | Print table of contents. |
| .TM | Thesis mode format. |
| .XS | Start table of contents entry. |
| .XE | End table of contents entry. |
| .XA | Additional table of contents entry. |
| .PX | Prints table of contents. |
| .XP | Exdented paragraph. |

Table 2-8 *New String Definitions*

| Definition | In `nroff` | In `troff` |
|---|---|---|
| `\*-` | Two dashes -- | Em dash — |
| `\*Q` | Open quote " | Open quote “ |
| `\*U` | Close quote " | Close quote ” |

Table 2-9 `-ms` *Macro Request Summary*

| Macro Request | Initial Value | Cause Break? | Explanation |
|---|---|---|---|
| `.1C` | yes | yes | One column format on a new page. |
| `.2C` | no | yes | Two column format. |
| `.AB` | no | yes | Begin abstract. |
| `.AE` | – | yes | End abstract. |
| `.AI` | no | yes | Author's institution follows. |
| `.AM` | – | no | New accent mark definitions |
| `.AT` | no | yes | Print '...Attached' and turn off line filling. |
| `.AU` | no | yes | Author's name follows. |
| `.B` *x* | no | no | Print *x* in boldface; if no argument switch to boldface. |
| `.B1` | no | yes | Begin text to be enclosed in a box. |
| `.B2` | no | yes | End text to be boxed and print it. |
| `.BT` | date | no | Bottom title, automatically invoked at foot of page. May be redefined. |
| `.BX` *x* | no | no | Print *x* in a box. |
| `.CM` | - | no | Cut mark between pages (only if troff). |
| `.CT` | - | yes | Chapter title in thesis mode only. Page number moved to CF. |
| `.DA` *x* | date | no | 'Date line' at bottom of page is *x* (only in `nroff`). Default is today. |
| `.DE` | – | yes | End displayed text. Implies `.KE`. |
| `.DS` *x* | no | yes | Start of displayed text to appear verbatim line–by–line. *x*=I for indented display (default), *x*=L for left-adjusted on the page, *x*=C for centered, *s*=B for make left–justified block, then center whole block. Implies `.KS`. |
| `.EF` *x* | – | no | Even three-part page footer *x* |
| `.EN` | – | yes | Space after equation produced by *eqn* or *neqn*. |

Table 2-9 -ms *Macro Request Summary— Continued*

| Macro Request | Initial Value | Cause Break? | Explanation |
|---|---|---|---|
| .EQ *x y* | – | yes | Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be I to indent equation (default), L to left–adjust the equation, or C to center it. |
| .FE | – | yes | End footnote. |
| .FS *x* | – | no | Start footnote. *x* is optional footnote label. The note will be printed at the bottom of the page. |
| .I *x* | no | no | Italicize *x*; if *x* is missing, italic text follows. |
| .IP *x y* | no | yes | Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5). |
| .KE | – | yes | End keep. Put kept text on next page if not enough room. |
| .KF | no | yes | Start floating keep. If the kept text must be moved to the next page, float later text back to this page. |
| .KS | no | yes | Start keeping following text. |
| .LG | no | yes | Make letters larger. |
| .LP | yes | yes | Start left-blocked paragraph. |
| .ND *date* | – | no | Use date supplied if any as page footer; only in special format positions. |
| .NH *n* | – | yes | Same as .SH with section number supplied automatically. Numbers are multilevel, like 1.2.3, where *n* tells what level is wanted (default is 1). |
| .NL | yes | no | Make letters normal size. |
| .IX *x y* | – | yes | Index entries *w* and *y* and so on up to 5 levels. Make letters normal size. |
| .OF *x* | – | no | Odd three-part page footer. |
| .OH *header* | – | no | Odd three-part page header. |
| .P1 | – | no | Print header on first page (only in thesis mode). |
| .PP | no | yes | Begin paragraph. First line indented. |
| .PT | pg # | – | Page title, automatically invoked at top of page. May be redefined. |

Table 2-9 -ms *Macro Request Summary— Continued*

| Macro Request | Initial Value | Cause Break? | Explanation |
|---|---|---|---|
| `.PX` *x* | – | yes | Print table of contents; *x*=no suppresses title. |
| `.QP` | – | yes | Begin single paragraph which is indented and shorter. |
| `.R` | yes | no | Roman text follows. |
| `.RE` | – | yes | End relative indent level. |
| `.RP` | no | – | Cover sheet and first page for released paper. Must precede other requests. |
| `.RS` | – | yes | Start level of relative indentation. Following `.IP`s are measured from current indentation. |
| `.SH` | – | yes | Section head follows, font automatically bold. |
| `.SM` | no | no | Make letters smaller. |
| `.TA` *x*... | 5... | no | Set tabs in ens. Default is 5 10 15 ... |
| `.TE` | – | yes | End table. |
| `.TH` | – | yes | End heading section of table. |
| `.TL` | no | yes | Title follows. |
| `.TM` | off | no | Thesis mode format. |
| `.TS` *x* | – | yes | Begin table; if *x* is H, table has repeated heading on subsequent pages. |
| `.UL` *x* | – | yes | Underline argument, even in `troff`. |
| `.XA` *x y* | – | yes | Another index entry; *x*=page for no for none, *y*=indent. |
| `.XE` | – | yes | End index entry or series of `.IX` entries. |
| `.XS` *x y* | – | yes | Begin index entry; *x*=page or no for none, *y*=indent. |
| `.UL` *x* | – | yes | Underline argument, even in `troff`. |

Table 2-10 -ms *String Definitions*

| Name | Definition | In nroff | In troff |
|---|---|---|---|
| quote | `\*Q` | " | “ |
| unquote | `\*U` | " | ” |
| dash | `\*-` | -- | — |
| month of year | `\*(MO` | March | March |
| current date | `\*(DY` | 6 March 1988 | 6 March 1988 |
| numbered footnote | `\**` | [1]*footnote* | $^1$*footnote* |

The following table summarizes command lines you use to print and display documents. Use the same order with `troff` for preprocessing files with `tbl` and `eqn`.

If you use the two-column `.2C` request, either pipe the `nroff` output through `col` or make the first line of the input `.pi /usr/bin/col`.

Table 2-11 *Printing and Displaying Documents*

| What You Want to Do | How to Do it |
|---|---|
| Display file on screen | **nroff -ms** *file(s)* **\| more** |
| Print file on line printer | **nroff -ms** *file(s)* **\| lpr** |
| Print file with tables | **tbl** *file(s)* **\| nroff -ms \| lpr** |
| Print file with equations | **neqn** *file(s)* **\| nroff -ms \| lpr** |
| Print file with both | **tbl** *file(s)* **\| neqn \| nroff -ms \| lpr** |
| Print file using `troff` | **troff -ms** *file(s)* **\| lpr -t** |

# 3

# The -man Macro Package

# 3

# The −man Macro Package

The −man macro package is used to format the manual pages to look like those in the *SunOS Reference Manual*, for example.

## 3.1. Parts of a Manual Page

A manual page consists of several parts:

- The first part is the *header and footer* or .TH line. This line identifies the manual page and sets up the titles and other information to print the page headers and footers.
- The next few sections are all introduced by .SH macro requests.

A skeleton command file would look something like this:

```
.TH XX 1 "7 November 1984"
.SH NAME
.SH SYNOPSIS
.SH DESCRIPTION
.SH OPTIONS
.SH FILES
.SH "SEE ALSO"
.SH DIAGNOSTICS
.SH BUGS
```

The sections have the following meanings:

| | |
|---|---|
| **NAME** | The name of the command and a short description. |
| **SYNOPSIS** | A short synopsis of the command including its options and arguments. |
| **DESCRIPTION** | A brief narrative description of what the command does. |
| **OPTIONS** | A list of the options in terse itemized list format. |
| **FILES** | Names of files that this command uses or creates. |
| **SEE ALSO** | Other relevant commands and files and manuals. |
| **BUGS** | Known deficiencies in the command. |

Occasionally there may be other sections you can add. For instance, a couple of the manual pages have a section called RESTRICTIONS, which contains the notice that this software is not distributed outside of the United States of America.

Leave out sections that do not apply — it is not necessary to have a title without any content to go with it. Definitely avoid sections that read:

BUGS
None.

## 3.2. Coding Conventions

The following subsections compose a fairly detailed description of what the different sections of the manual page contain.

### The Header and Footer Line (.TH) — Identifying the Page

The .TH macro is the macro that identifies the page. The format is

```
.TH n c x v m
```

This means, for example: Begin page named *n* of chapter *c*. The *x* argument is for extra commentary for the center page footer. The *v* argument alters the left portion of the page footer. The *m* argument alters the center portion of the page header. The .TH command line also incidentally sets the prevailing indent and tabs to .5i.

To code a manual page called `troff(1)`, for example, you would code a .TH macro like:

```
.TH TROFF 1 "today's date"
```

The third parameter to the .TH macro is the date on which you created or last changed the manual page. You code *today's date* in international form:

*numerical day spelled-out month numerical year*

So if today is September 3rd, 1984, you code the **.TH** macro like:

```
.TH TROFF 1 "3 September 1984"
```

### The NAME Line

The **NAME** line is a one-liner that identifies the command or program. You code the information like this:

```
.SH NAME
troff \- typeset or format documents
```

This line must be typed all in the Roman font with no font changes or point-size changes or any other text manipulation. Typing the command line all in Roman with no text manipulation is for the permuted index generator. It gets all confused if there is anything in that line other than plain text.

Note the \- in there — why do we type a \-? Well, in `troff` jargon, a simple - sign gets you a hyphen. We actually would like a en-dash (like –) instead of a hyphen, in lieu of actually having a em-dash (like —). This use of the \- to get a – is a UNIX† tradition.

### The SYNOPSIS Section

The **SYNOPSIS** line(s) show the user what options and arguments can be typed. The conventions for the `SYNOPSIS` have varied wildly over the years. Nonetheless, here are the guidelines:

- *Literal text* (that is, what the user types) is coded in **boldface**.
- *Variables* (that is, things someone might *substitute for*) are typed in *italic text.*
- *Optional things* are enclosed in brackets — that is the characters `[` and `]`.
- *Alternatives* are separated by the vertical bar sign ( | ).

The synopsis should show what the options are — some manual pages used to read like this:

> SYNOPSIS
>     **troff** [ *options* ] filename ...

but it *should* read:

> SYNOPSIS
>     **troff** [**–o***pagelist* ] [**–n***N* ] [**–m** *name* ] ... [ filename ]

### The DESCRIPTION Section

The **DESCRIPTION** section of a manual page should contain a brief description of what the command does *for the user*, in terms that the user cares about.

Within the `DESCRIPTION` and `OPTIONS` sections, *italic text* is used for filenames and command names. The rationale here is that UNIX commands are simply files. When referring to other manual pages, you type the name in italics and the following parenthesized section number in Roman, as in `make(1)`. Use the `-man` macro `.IR` to get alternating words joined in italic and Roman fonts. Note that the macros that join alternating words in different fonts (`.IR`, `.IB`, `.BR`, `.BI`, `.RI`, `.RB`) all accept only *six* parameters. See the section on how to format a manual page for more formatting rules.

Part of the description in the `grep` manual page used to read:

..... `grep` patterns are limited regular expressions in the style of `ed(1)`; it uses a compact nondeterministic algorithm. `egrep` patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. `fgrep` patterns are fixed strings; it is fast and compact.

Most users do not care that `egrep` uses a fast deterministic algorithm. As an example of a more useful way of describing a command for the user, here is how

† UNIX is a registered trademark of AT&T.

that sentence in the `grep` manual page currently reads.

..... `grep` patterns are limited regular expressions in the style of `ed(1)`. `egrep` patterns are full regular expressions including alternation. `fgrep` searches for lines that contain one of the (newline-separated) *strings*. `fgrep` patterns are fixed strings — no regular expression metacharacters are supported.

Here's another bad example: the `lpr(1)` command used to tell you that the –s option uses the `symlink` (2) system call to make a symbolic link to the data file instead of copying the data file to the spool area. The user may not know what this means or how to use the information. The description was changed to just tell you that the –s option makes a symbolic link to the data file. How it is done is of little concern to some poor blighter who just wants to print a file.

### The OPTIONS Section

The **OPTIONS** section of a manual page contains an itemized list of the options that the command recognizes, and how the options affect the behavior of the command. The general format for this section is

**–option** Description of what the option does.

A specific example from the `troff` manual page looks like this:

OPTIONS

Options may appear in any order as long as they appear **before** the files.

**–o***list*

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

**–n***N*

Number first generated page *N*.

**–m***name*

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

**–r***aN*

Set register *a* (one-character) to *N*.

**–i**

Read standard input after the input files are exhausted.

**–q**

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

**–t**

Direct output to the standard output instead of the phototypesetter. In general, you will have to use this option if you don't have a typesetter attached to the system.

**–a**

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

Some options of `troff` only apply if you have a C/A/T typesetter attached to your system. These options are here for historical reasons:

–s*N*

Stop every *N* pages. `troff` stops the phototypesetter every *N* pages, produces a trailer to allow changing cassettes, and resumes when the typesetter's start button is pressed.

**–f**

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

**–w**

Wait until phototypesetter is available, if currently busy.

**-b**

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

–p*N*

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

## The FILES Section

The **FILES** section of a manual page contains a list of the files that the program accesses, creates, or modifies. Obviously, you can leave this section out if the program uses no files.

The example from the `troff` manual page looks like this:

> If the file `/usr/adm/tracct` is writable, `troff` keeps phototypesetter accounting records there. The integrity of that file may be secured by making `troff` a 'set user-id' program.

**FILES**

| | |
|---|---|
| /tmp/ta* | temporary file |
| /usr/lib/tmac/tmac.* | standard macro files |
| /usr/lib/term/* | terminal driving tables for nroff |
| /usr/lib/font/* | font width tables for troff |
| /dev/cat | phototypesetter |
| /usr/adm/tracct | accounting statistics for /dev/cat |

### The SEE ALSO Section

The **SEE ALSO** section of a manual page contains a list of references to other programs, files, and manuals relating to this program. For example, on the `troff` manual page, the **SEE ALSO** section looks like this:

> SEE ALSO
> *Formatting Documents* and *Using NROFF and TROFF*,
> `nroff(1),eqn(1),tbl(1),ms(7),me(7),`
> `man(7),col(1)`

Make sure that the references are useful — the `rm(1)` command references the `unlink(2)` system call. Does the user care what system call is used to get rid of a file? It's not intuitive that you use a function called `unlink` to remove a file.

Leave this section out if there are no interesting references.

### The BUGS Section

The **BUGS** section of a manual page is to convey limitations or bad behavior of the command to the reader. Please limit bugs to these categories.

Leave this section out altogether if there are no bugs worth noting.

## 3.3. New Features of the -`man` Macro Package

### New Number Registers

Recent enhancements to the -`man` macro package facilitate including manual pages in manuals. The major new features are number registers that can be set from the `itroff`, `iroff`, `troff`, `ditroff`, or `nroff` command line. The number registers are:

**D** Format the document for double-sided printing if the **D** number register is set to 1. Double-sided printing means that the page numbers appear in different locations on odd and even pages. Page numbers appear in the running footers in the lower *right* corner of *odd-numbered* pages and in the lower *left* corner of *even-numbered* pages.

**C** Number pages contiguously — pages are numbered 1, 2, 3,... even when you format more than one manual page at a time. Every new topic used to start numbering at page 1.

**P***nnn* Start Page numbering at page *nnn* — page numbering starts at page 1 if not otherwise specified.

**X***nnn* Number pages as *nnn***a**, *nnn***b**, etc when the current page number becomes *nnn*. This feature is for generating update pages to slot in between existing pages. For example, if a new page called *skyversion*(8) should be included in an interim release, we can number that page as page '26a' and drop it into the existing manual in a reasonable fashion.

## Using the Number Registers

Number registers are set from the `itroff`, `iroff`, `troff`, `ditroff`, or `nroff` command line by the **–r** (set register) option, followed immediately by the *one-letter* name of the register, followed immediately by the value to put into the number register:

```
hostname% troff -man -rD1 manpage.1
hostname%
```

This example shows how to request a format suitable for double-sided printing.

If your `grab(1)` manual page used to be three pages long and is now five pages long, you need the pages numbered 1, 2, 3, 3a, and 3b instead of 1, 2, 3, 4, and 5. You get this effect by using the **–rX** option on the command line, setting the **X** register to 3:

```
hostname% troff -man -rX3 grab.1
hostname%
```

We introduced the `screendump(1)` and `screenload(1)` manual pages in the 1.2 release. `screendump(1)` and `screenload(1)` come immediately after the `sccsdiff(1)` manual page. `sccsdiff`'s last page number is page 260, so we get `screendump(1)` and `screenload(1)` formatted with this command to start page numbering at 260 and to start putting in extra page letters at 260 as well:

```
hostname% troff -man -rP260 -rX260 screendump.1 screenload.1
hostname%
```

## 3.4. How to Format a Manual Page

Any text argument *t* to a macro request may be from zero to six words. Quotes my be used to include blanks in a 'word'. If the text field is empty, the macro request is applied to the next input line with text to be printed. In this way, **.I** italicizes an entire line, and **.SM** followed on a separate line by **.B** creates small, bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to the default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to the default values before each paragraph, and after processing font- and size-setting macros.

These strings are predefined by `-man`:

| | |
|---|---|
| *R | ®, '(Reg)' in `nroff`. |
| *S | Change to default type size. |

Table 3-1 *Summary of the* `-man` *Macro Requests*

| ***Request*** | ***Cause Break*** | ***If no Argument*** | ***Explanation*** |
|---|---|---|---|
| `.B` *t* | no | *t*=next text line | Text *t* is bold. |
| `.BI` *t* | no | *t*=next text line | Join words of *t* alternating bold and italic. |
| `.BR` *t* | no | *t*=next text line | Join words of *t* alternating bold and Roman. |
| `.DT` | no | .5i 1i... | Restore default tabs. |
| `.HP` *i* | yes | *i*=prevailing indent | Set prevailing indent to *i*. Begin paragraph with hanging indent. |
| `.I` *t* | no | *t*=next text line | Text *t* is italic. |
| `.IB` *t* | no | *t*=next text line | Join words of *t*alternating italic and bold. |
| `.IP` *x i* | yes | *x*="" | Same as .TP with tag *x*. |
| `.IR` *t* | no | *t*=next text line | Join words of *t* alternating italic and Roman. |
| `.LP` | yes | | Same as .PP. |
| `.PD` *d* | no | *d*=.4v | Interparagraph distance is *d*. |
| `.PP` | yes | | Begin paragraph. Set prevailing indent to .5i. |
| `.RE` | yes | | End of relative indent. Set prevailing indent to amount of starting .RS. |
| `.RB` *t* | no | *t*=next text line | Join words of *t* alternating Roman and bold. |
| `.RI` *t* | no | *t*=next text line | Join words of *t* alternating Roman and italic. |
| `.RS` *i* | yes | *i*=prevailing indent | Start relative indent, move left margin in distance *i*. Set prevailing indent to .5i for nested indents. |
| `.SB` *t* | no | | Print *t* in smaller boldface. |
| `.SH` *t* | yes | *t*=next text line | Subheading. |
| `.SM` *t* | yes | *t*=next text line | Text *t* is two point sizes smaller than surrounding text. |
| `.TH` *n c x v m* | yes | | Begin page named *n* of chapter *c*. The *x* argument is for extra commentary for the center page footer. The *v* argument alters the left portion of the page footer. The *m* argument alters the center portion of the page header. The `.TH` command line also incidentally sets the prevailing indent and tabs to .5i. |
| `.TP` *i* | yes | *i*=prevailing indent | Set the prevailing indent to *i*. Begin indented paragraph with hanging tag given by the next text line. If the tag does not fit, place it on a separate line. |
| `.TX` *tp* | no | | Resolve the title abbreviation *t*; join to punctuation *p*. |

To learn how to format manual pages on your terminal or workstation screen, refer to the `man(1)` manual page.

# 4

# Formatting Documents with the -me Macros

# 4

# Formatting Documents with the -me Macros

This chapter describes the -me macro package.[1] The first part of each section presents the material in user's guide format and the second part lists the macro requests for quick reference. The chapter contents include descriptions of the basic requests, displays, annotations, such as footnotes, and how to use -me with nroff and troff.

We assume that you are somewhat familiar with nroff and troff and that you know something about breaks, fonts, point sizes, the use and definition of number registers and strings, and scaling factors for ens, points, vertical line spaces, etc. If you are a newcomer, try out the basic features as you read along.

All request names in -me follow a naming convention. You may define number registers, strings, and macros, provided that you use single-character, upper case names or double-character names consisting of letters and digits with at least one upper case letter. Do not use special characters in the names you define. The word *argument* in this chapter means a word or number which appears on the same line as a request and which modifies the meaning of that request. Default parameter values are given in brackets. For example, the request

```
.sp
```

spaces one line, and

```
.sp 4
```

spaces four lines. The number '4' is an *argument* to the .sp request; it modifies .sp to produce four lines instead of one. Spaces separate arguments from the request and from each other.

[1] The material in this chapter is derived from *Writing Papers with* nroff *Using* -me, E. P. Allman, and -me *Reference Manual*, E. P. Allman, University of California, Berkeley.

## 4.1. Using -me

When you have your raw text ready, run the nroff formatter with the -me option to send the output to the standard output, your workstation screen. Type:

```
hostname% nroff -me -Ttype files
hostname%
```

where *type* describes the type of terminal you are outputting to. Common values are dtc for a DTC 300s (daisy-wheel type) printer and lpr for the line printer. If you omit the -T flag, a 'lowest common denominator' terminal is assumed; this is good for previewing output on most terminals.

For easier viewing, pipe the output to more or redirect it to another file.

For formatting on the phototypesetter with troff (or your installation's equivalent), use:

```
hostname% troff -me file
hostname%
```

## 4.2. Basic -me Requests

The following sections provide descriptions and examples of the basic -me requests.

### Paragraphs

The -me package has requests for formatting standard, left block, and indented paragraphs.

Standard Paragraph — .pp

Begin standard paragraphs by using the .pp request. For example, the input:

```
.pp
Now is the time for all good men
to come to the aid of their party.
Four score and seven years ago,...
```

produces

    Now is the time for all good men to come to the aid of their party. Four score and seven years ago,...

that is, a blank line followed by an indented first line.

Do not begin the sentences of a paragraph with a space, since blank lines and lines beginning with spaces cause a break. For example, if you type:

```
.pp
Now is the time for all good men
      to come to the aid of their party.
Four score and seven years ago,...
```

The output is:

Now is the time for all good men
to come to the aid of their party. Four score and seven years ago,...

A new line begins after the word 'men' because the second line begins with a space character.

Because the first call to one of the paragraph macros defined in a section or the .H macro *initializes* the macro processor, do not use any of the following requests: .sc, .lo, .th, or .ac (see the section called "Section Headings"). Also, avoid changing parameters, notably page length and header and footer margins, which have a global effect on the format of the page.

### Left Block Paragraphs — .lp

A formatted paragraph can start with a blank line and with the first line indented. You can get left-justified block-style paragraphs as shown throughout this manual by using .lp (left paragraph) instead of .pp.

### Indented Paragraphs — .ip and .np

Sometimes you want to use paragraphs that have the *body* indented, and the first line exdented, that is, the opposite of indented, with a label. Use the .ip request for this. A word specified on the same line as .ip is printed in the margin, and the body is lined up at a specified position. For example, the input:

```
.ip one
This is the first paragraph.
Notice how the first line
of the resulting paragraph lines up
with the other lines in the paragraph.
.ip two
And here we are at the second paragraph already.
You may notice that the argument to .ip
appears in the margin.
.lp
We can continue text...
```

produces as output:

one This is the first paragraph. Notice how the first line of the resulting paragraph lines up with the other lines in the paragraph.

two And here we are at the second paragraph already. You may notice that the argument to .ip appears in the margin.

We can continue text without starting a new indented paragraph by using the .lp request.

If you have spaces in the label of an .ip request, use an *unpaddable space* instead of a regular space. This is typed as a backslash character (\) followed by a space. For example, to print the label 'Part 1', type:

```
.ip "Part\ 1"
```

If a label of an indented paragraph, that is, the argument to .ip, is longer than the space allocated for the label, .ip begins a new line after the label. For example, the input:

```
.ip longlabel
This paragraph has a long label.
The first character of text on the first line will not
line up with the text on second and subsequent lines,
although they will line up with each other.
```

produces:

longlabel

This paragraph has a long label. The first character of text on the first line will not line up with the text on second and subsequent lines, although they will line up with each other.

You can change the size of the label by using a second argument which is the size of the label. For example, you can produce the above example correctly by saying:

```
.ip longlabel 10
```

which will make the paragraph indent 10 spaces for this paragraph only. For example:

longlabel

This paragraph has a long label. The first character of text on the first line will not line up with the text on second and subsequent lines, although they will line up with each other.

If you have many paragraphs to indent all the same amount, use the *number register* `ii`. For example, to leave one inch of space for the label, type:

```
.nr ii 1i
```

somewhere before the first call to `.ip`.

If you use `.ip` without an argument, no hanging tag is printed. For example, the input:

```
.ip [a]
This is the first paragraph of the example.
We have seen this sort of example before.
.ip
This paragraph is lined up with the previous paragraph,
but it does not have a tag in the margin.
```

produces as output:

[a] This is the first paragraph of the example. We have seen this sort of example before.

This paragraph is lined up with the previous paragraph, but it does not have a tag in the margin.

A special case of `.ip` is `.np`, which automatically numbers paragraphs sequentially from 1. The numbering is reset at the next `.pp`, `.lp`, or `.H` request. For example, the input:

```
.np
This is the first point.
.np
This is the second point.
Points are just regular paragraphs
which are given sequence numbers automatically
by the .np request.
.lp
This paragraph will reset numbering by .np.
.np
For example,
we have reverted to numbering from one now.
```

generates:

(1) This is the first point.

(2) This is the second point. Points are just regular paragraphs which are given sequence numbers automatically by the `.np` request.

This paragraph will reset numbering by `.np`.

(1) For example, we have reverted to numbering from one now.

## Paragraph Reference

`.lp` Begin left-justified paragraph. Centering and underlining are turned off if they were on, the font is set to `\n(pf` [1], the type size is set to `\n(pp` [10p], and a `\(nps` space is inserted before the paragraph (0.35v in `troff`, 1v or 0.5v in `nroff` depending on device resolution). The indent is reset to `\n($1` [0] plus `\n(po` [0] unless the paragraph is inside a display (see `.ba` in "Miscellaneous Requests"). At least the first two lines of the paragraph are kept together on a page.

`.pp` Like `.lp`, except that it puts `\n(pi` [5n] units of indent. This is the standard paragraph macro.

`.ip` *T I* Indented paragraph with hanging tag. The body of the following paragraph is indented *I* spaces (or `\n(ii` [5n] spaces if *I* is not specified) more than a non-indented paragraph is (such as with `.lp`). The title *T* is exdented. The result is a paragraph with an even left edge and *T* printed in the margin. Any spaces in *T* must be unpaddable. If *T* will not fit in the space provided, `.ip` starts a new line.

`.np` An `.ip` variant that numbers paragraphs. Numbering is reset after an `.lp`, `.pp`, or `.H`. The current paragraph number is in `\n$p`.

## 4.3. Headers and Footers — `.he` and `.fo`

You can put arbitrary headers and footers at the top and bottom of every page. Two requests of the form `.he` *title* and `.fo` *title*' define the titles to put at the head and the foot of every page, respectively. The titles are called *three-part* titles, that is, there is a left-justified part, a centered part, and a right-justified part. The first character of *title* (whatever it may be) is used as a delimiter to separate these three parts. You can use any character but avoid the backslash and double quote marks. The percent sign is replaced by the current page number whenever it is found in the title. For example, the input:

```
.he '' %''
.fo 'Jane Jones'' My Book'
```

results in the page number centered at the top of each page, 'Jane Jones' in the lower left corner, and 'My Book' in the lower right corner.

If there are two blanks adjacent anywhere in the title or more than eight blanks total, you must enclose three-part titles in single quotes.

Headers and footers are set in font `\n(tf` [3] and size `\n(tp` [10p]. Each of the definitions applies as of the *next* page.

Three number registers control the spacing of headers and footers. `\n(hm` [4v] is the distance from the top of the page to the top of the header, `\n(fm` [3v] is the distance from the bottom of the page to the bottom of the footer, `\n(tm` [7v] is the distance from the top of the page to the top of the text, and `\n(bm` [6v] is the distance from the bottom of the page to the bottom of the text (nominal). You can also specify the space between the top of the page and the header, the header and the first line of text, the bottom of the text and the footer, and the footer and the bottom of the page with the macros `.m1`, `.m2`, `.m3`, and `.m4`.

### Headers and Footers Reference

`.he` *'l'm'r'* Define three-part header, to be printed on the top of every page.

`.fo` *'l'm'r'* Define footer, to be printed at the bottom of every page.

`.eh` *'l'm'r'* Define header, to be printed at the top of every even-numbered page.

`.oh` *'l'm'r'* Define header, to be printed at the top of every odd-numbered page.

`.ef` *'l'm'r'* Define footer, to be printed at the bottom of every even-numbered page.

`.of` *'l'm'r'* Define footer, to be printed at the bottom of every odd-numbered page.

`.hx` Suppress headers and footers on the next page.

`.m1` +*N* Set the space between the top of the page and the header [4v].

`.m2 +`*N* Set the space between the header and the first line of text [2v].

`.m3 +`*N* Set the space between the bottom of the text and the footer [2v].

`.m4 +`*N* Set the space between the footer and the bottom of the page [4v].

`.ep` End this page, but do not begin the next page. Useful for forcing out footnotes. Must be followed by a `.bp` or the end of input.

`.$h` Called at every page to print the header. May be redefined to provide fancy headers, such as, multi-line, but doing so loses the function of the `.he`, `.fo`, `.eh`, `.oh`, `.ef`, and `.of` requests, as well as the chapter-style title feature of `.+c`.

`.$f` Print footer; same comments apply as in `.$h`.

`.$H` A normally undefined macro which is called at the top of each page after processing the header, initial saved floating keeps, etc.; in other words, this macro is called immediately before printing text on a page. Used for column headings and the like.

## Double Spacing — `.ls 2`

`nroff` will double space output text automatically if you use the request

`.ls 2`, as is done in this section. You can revert to single-space mode by typ-

ing `.ls 1`.

## Page Layout

You can change the way the printed copy looks, sometimes called the *layout* of the output page with the following requests. Most of these requests adjust the placing of 'white space' (blank lines or spaces). In these explanations, replace characters in italics with values you wish to use; bold characters represent characters which you should actually type.

Use `.bp` (break page) to start a new page.

The request `.sp` *N* leaves *N* lines of blank space. You can omit *N* to skip a single line or you can use the form *N* `i` (for *N* inches) or *N* `c` (for *N* centimeters). For example, the input:

```
.sp 1.5i
My thoughts on the subject
.sp
```

leaves one and a half inches of space, followed by the line 'My thoughts on the subject', followed by a single blank line.

The `.in` +*N* (indent) request changes the amount of white space on the left of the page. The argument *N* can be of the form + *N* (meaning leave *N* spaces more than you are already leaving), – *N*' (meaning leave *N* spaces less than you do now), or just *N* (meaning leave exactly *N* spaces). *N* can be of the form *N* `i` or *N* `c` also. For example, the input:

```
initial text
.in 5
some text
.in +1i
more text
.in −2c
final text
```

produces 'some text' indented exactly five spaces from the left margin, 'more text' indented five spaces plus one inch from the left margin (fifteen spaces on a pica typewriter), and 'final text' indented five spaces plus one inch minus two centimeters from the margin. That is, the output is:

```
initial text
          some text
                         more text
             final text
```

The `.ti` *+N* (temporary indent) request is used like `.in` *+N* when the indent should apply to one line only, after which it should revert to the previous indent. For example, the input:

```
.in 1i
.ti 0
Ware, James R.  The Best of Confucius,
Halcyon House, 1950.
An excellent book containing translations of
most of Confucius′ most delightful sayings.
A definite must for anyone interested in the
early foundations of Chinese philosophy.
```

produces:

Ware, James R. The Best of Confucius, Halcyon House, 1950. An excellent book containing translations of most of Confucius' most delightful sayings. A definite must for anyone interested in the early foundations of Chinese philosophy.

You can center text lines with the `.ce` (center) request. The line after the `.ce` is centered horizontally on the page. To center more than one line, use `.ce` *N*, where *N* is the number of lines to center, followed by the *N* lines. If you want to center many lines but don't want to count them, type:

```
.ce 1000
lines to center
.ce 0
```

The `.ce 0` request tells `nroff` to center zero more lines, in other words, to stop centering.

All of these requests cause a break; that is, they always start a new line. If you want to start a new line without performing any other action, use `.br` (break).

## Underlining — `.ul`

Use the `.ul` (underline) request to underline text. The `.ul` request operates on the next input line when it is processed. You can underline multiple lines by stating a count of *input* lines to underline, followed by those lines, the same as with the `.ce` request. For example, the input:

```
.ul 2
The quick brown fox
jumped over the lazy dog.
```

underlines those words in `nroff`. In `troff` they are italicized.

## Displays

Use displays to set off sections of text from the body of the paper. Major quotes, tables, and figures are types of displays, as are all the examples used in this manual. All displays except centered text blocks are single-spaced.

### Major Quotes — `.(q` and `.)q`

Major quotes are quotes which are several lines long, and hence are set in from the rest of the text without quote marks around them. Use `.(q` and `.)q` to surround the quote. For example, the input:

```
As Weizenbaum points out:
.(q
It is said that to explain is to explain away.
This maxim is nowhere so well fulfilled
as in the areas of computer programming,...
.)q
```

generates as output:

As Weizenbaum points out:

> It is said that to explain is to explain away. This maxim is nowhere so well fulfilled as in the areas of computer programming,...

### Lists — `.(l` and `.)l`

A *list* is an indented, single-spaced, unfilled display. You should use lists when the material to be printed should not be filled and justified like normal text. This is useful for columns of figures, for example. Surround the list text by the requests `.(l` and `.)l`. For example, type:

```
Alternatives to avoid deadlock are:
.(l
Lock in a specified order
Detect deadlock and back out one process
Lock all resources needed before proceeding
.)l
```

to produce:

Alternatives to avoid deadlock are:
        Lock in a specified order
        Detect deadlock and back out one process
        Lock all resources needed before proceeding

Keeps — `.(b` and `.)b`, `.(z` and `.)z`

A *keep* is a group of lines that are kept together on a single page. If less vertical space exists on the current page than can accommodate text within a keep, the formatter begins a new page and keeps those lines together. Keeps are useful for printing diagrams because you don't want them spread across two pages. For comparison, lists may be broken over a page boundary, whereas keeps may not.

Blocks are the basic kind of keep. They begin with the request `.(b` and end with the request `.)b`. If there is not enough room on the current page for everything in the block, the formatter begins a new page. This has the unaesthetic effect of leaving blank space at the bottom of the page. When this is not appropriate, you can use the alternative called a *floating keep*.

*Floating keeps* move relative to the text. Hence, they are good for things which will be referred to by name, such as 'See figure 3'. A floating keep will appear at the bottom of the current page if it will fit; otherwise, it will appear at the top of the next page. Floating keeps begin with the line `.(z` and end with the line `.)z`. An example of a floating keep is:

```
.(z
.hl
Text of keep to be floated.
.sp
.ce
Figure 1.  Example of a Floating Keep.
.hl
.)z
```

The `.hl` request draws a horizontal line so the figure stands out from the text.

## 4.4. Fancy Displays

Keeps and lists are normally collected in *nofill* mode, so they are good for tables and displays. If you want a display in fill mode (for text), type `.(l F`. Throughout this section, comments applied to `.(l` also apply to `.(b` and `.(z`. This kind of display produced by `.(l` is indented from both margins. For example, the input:

```
.(l F
And now boys and girls,
a newer, bigger, better toy than ever before!
Be the first on your block to have your own computer!
Yes kids, you too can have one of these modern
data processing devices.
You too can produce beautifully formatted papers
without even batting an eye!
.)l
```

will be formatted as:

> And now boys and girls, a newer, bigger, better toy than ever before! Be the first on your block to have your own computer! Yes kids, you too can have one of these modern data processing devices. You too can produce beautifully formatted papers without even batting an eye!

Lists and blocks are also normally indented, while floating keeps are normally left-justified. To get a left-justified list, type `.(l L`. To center a list line-for-line, type `.(l C`. For example, to get a filled, left-justified list, use:

```
.(l L F
text of block
.)l
```

The input:

```
.(l
first line of unfilled display
more lines
.)l
```

produces the indented text:

> first line of unfilled display
> more lines

Typing the character `L` after the `.(l` request produces the left-justified result:

first line of unfilled display
more lines

Using `C` instead of `L` produces the line-at-a-time centered output:

first line of unfilled display
more lines

Sometimes you may want to center several lines as a group, rather than centering them one line at a time. To do this use centered blocks, which are surrounded by the requests `.(c` and `.)c`. All the lines are centered as a unit, such that the longest line is centered, and the rest are lined up around that line. Notice that lines do not move relative to each other using centered blocks, whereas they do using the `C` keep argument.

Centered blocks are *not* keeps, and you may use them in conjunction with keeps. For example, to center a group of lines as a unit and keep them on one page, use:

```
.(b L
.(c
first line of unfilled display
more lines
.)c
.)b
```

to produce:

first line of unfilled display
more lines

the result would have been the same, but with no guarantee that the lines of the centered block would have all been on one page. Note the use of the `L` argument to `.(b`; this centers the centered block within the entire line rather than within the line minus the indent. Also, you must nest the center requests *inside* the keep requests.

**Display Reference**

All displays except centered blocks and block quotes are preceded and followed by an extra `\n(bs` (same as `\n(ps`) space. Quote spacing is stored in a separate register; centered blocks have no default initial or trailing space. The vertical spacing of all displays except quotes and centered blocks is stored in register `\n($R` instead of `\n($r`.

`.(l` *mf* Begin list. Lists are single-spaced, unfilled text. If *f* is `F`, the list will be filled. If *m* `[I]` is `I` the list is indented by `\n(bi` [4n]; if it is `M`, the list is indented to the left margin; if it is `L`, the list is left-justified with respect to the text (different from `M` only if the base indent (stored in `\n($i` and set with `.ba`) is not zero); and if it is `C`, the list is centered on a line-by-line basis. The list is set in font `\n(df` [0]. You must use a matching `.)l` to end the list. This macro is almost like `.DS` except that no attempt is made to keep the display on one page.

`.)l` End list.

`.(q` Begin major quote. The lines are single-spaced, filled, moved in from the main body of text on both sides by `\n(qi` [4n], preceded and followed by `\n(qs` (same as `\n(bs`) space, and are set in point size `\n(qp`, that is, one point smaller than the surrounding text.

`.)q` End major quote.

`.(b` *mf* Begin block. Blocks are a form of *keep*, where the text of a keep is kept together on one page if possible. Keeps are useful for tables and figures which should not be broken over a page. If the block will not fit on the current page a new page is begun, unless that would leave more than `\n(bt` [0] white space at the bottom of the text. If `\n(bt` is zero, the threshold feature is turned off. Blocks are not filled unless *f* is `F`, when they are filled. The block will be left-justified if *m* is `L`, indented by `\n(bi` [4n] if *m* is `I` or absent, centered (line-for-line) if *m* is `C`, and left justified to the margin, not to the base indent, if *m* is `M`. The block is set in font `\n(df` [0].

`.)b` End block.

`.(z` *mf* Begin floating keep. Like `.(b` except that the keep is *floated* to the bottom of the page or the top of the next page. Therefore, its position relative to the text changes. The floating keep is preceded and followed by `\n(zs` [1v] space. Also, it defaults to mode `M`.

`.)z` End floating keep.

`.(c` Begin centered block. The next keep is centered as a block, rather than on a line-by-line basis as with `.(b C`. This call may be nested inside keeps.

`.)c` End centered block.

## Annotations

There are a number of requests to save text for later printing. *Footnotes* are printed at the bottom of the current page. *Delayed text* is intended to be a variant form of footnote; the text is printed only when explicitly called for, such as at the end of each chapter. *Indexes* are a type of delayed text having a tag, usually the page number, attached to each entry after a row of dots. Indexes are also saved until explicitly called for.

### Footnotes — .(f and .)f

Footnotes begin with the request .(f and end with the request .)f. The current footnote number is maintained automatically, and can be used by typing **, to produce a footnote number.[2] The number is automatically incremented after every footnote. For example, the input:

```
.(q
A man who is not upright
and at the same time is presumptuous;
one who is not diligent and at the same time is ignorant;
one who is untruthful and at the same time is incompetent;
such men I do not count among acquaintances.\**
.(f
\**James R. Ware,
.ul
The Best of Confucius,
Halcyon House, 1950.
Page 77.
.)f
.)q
```

generates the result:

> A man who is not upright and at the same time is presumptuous; one who is not diligent and at the same time is ignorant; one who is untruthful and at the same time is incompetent; such men I do not count among acquaintances.[3]

Make sure that the footnote appears *inside* the quote, so that the footnote will appear on the same page as the quote.

### Delayed Text

Delayed text is very similar to a footnote except that it is printed when explicitly called for. Use this feature to put a list of references at the end of each chapter, as is the convention in some disciplines. Use *# on delayed text instead of ** as on footnotes.

If you are using delayed text as your standard reference mechanism, you can still use footnotes, except that you may want to refer to them with special characters* rather than numbers.

### Indexes — .(x .)x and .xp

An *index* resembles delayed text, in that it is saved until called for. It is actually more like a table of contents, since the entries are not sorted alphabetically. However, each entry has the page number or some other tag appended to the last line of the index entry after a row of dots.

Index entries begin with the request .(x and end with .)x. An argument to the .)x indicates the value to print as the 'page number.' It defaults to the current page number. If the page number given is an underscore (_), no page number or

[2] Like this.

[3] James R. Ware, *The Best of Confucius*, Halcyon House, 1950. Page 77.

* Such as an asterisk.

line of dots is printed at all. To get the line of dots without a page number, type **.)x ""**, which specifies an explicitly null page number.

The `.xp` request prints the index.

For example, the input:

```
.(x
Sealing wax
.)x 9
.(x
Cabbages and kings
.xp
```

generates:

```
Sealing wax ..................................................................................................        9
Cabbages and kings
        < etc. >
```

The `.(x` request may have a single-character argument, specifying the *name* of the index; the normal index is **x**. Thus, you can maintain several *indices* simultaneously, such as a list of tables and a table of contents.

Notice that the index must be printed at the *end* of the paper, rather than at the beginning where it will probably appear (as a table of contents); you may have to rearrange the pages after printing.

## Annotations Reference

`.(d` Begin delayed text. Everything in the next keep is saved for output later with `.pd` in a manner similar to footnotes.

`.)d` *n* End delayed text. The delayed text number register `\n($d` and the associated string `\*#` are incremented if `\*#` has been referenced.

`.pd` Print delayed text. Everything diverted via `.(d` is printed and truncated. You might use this at the end of each chapter.

`.(f` Begin footnote. The text of the footnote is floated to the bottom of the page and set in font `\n(ff` [1] and size `\n(fp` [8p]. Each entry is preceded by `\n(fs` [0.2v] space, is indented `\n(fi` [3n] on the first line, and is indented `\n(fu` [0] from the right margin. Footnotes line up underneath two-column output. If the text of the footnote will not all fit on one page, it will be carried over to the next page.

`.)f` *n* End footnote. The number register `\n($f` and the associated string `\**` are incremented if they have been referenced.

`.$s` The macro to generate the footnote separator. You may redefine this macro to give other size lines or other types of separators. It currently draws a 1.5-inch line.

`.(x` *x* Begin index entry. Index entries are saved in the index *x* until called up with `.xp`. Each entry is preceded by a `\n(xs` [0.2v] space. Each entry is 'undented' by `\n(xu` [0.5i]; this register tells how far the page number extends into the right margin.

`.)x` *P A* End index entry. The index entry is finished with a row of dots with *A* [null] right justified on the last line, such as for an author's name, followed by *P* [`\n%` ]. If *A* is specified, *P* must be specified; `\n%` can be used to print the current page number. If *P* is an underscore, no page number and no row of dots are printed.

`.xp` *x* Print index *x* [x]. The index is formatted in the font, size, and so forth in effect at the time it is printed, rather than at the time it is collected.

## 4.5. Fancy Features

A large number of fancier requests exist, notably requests to provide other sorts of paragraphs, numbered sections of the form '1.2.3', such as those used in this manual, and multicolumn output.

### Section Headings — `.sh` and `.uh`

You can automatically generate section numbers, using the `.sh` request. You must tell `.sh` the *depth* of the section number and a section title. The depth specifies how many numbers separated by decimal points are to appear in the section number. For example, the section number '4.2.5' has a depth of three.

Section numbers are incremented if you add a number. Hence, you increase the depth, and the new number starts out at one. If you subtract section numbers, or keep the same number, the final number is incremented. For example, the input:

```
.sh 1 "The Preprocessor"
.sh 2 "Basic Concepts"
.sh 2 "Control Inputs"
.sh 3
.sh 3
.sh 1 "Code Generation"
.sh 3
```

produces as output the result:

```
1. The Preprocessor
1.1. Basic Concepts
1.2. Control Inputs
1.2.1.
1.2.2.
2. Code Generation
2.1.1.
```

You can specify the beginning section number by placing the section number after the section title, using spaces instead of dots. For example, the request:

```
.sh 3 "Another section" 7 3 4
```

will begin the section numbered '7.3.4'; all subsequent `.sh` requests will be numbered relative to this number.

There are more complex features which indent each section proportionally to the depth of the section. For example, if you type:

```
.nr si Nx
```

each section will be indented by an amount *N*. *N* must have a scaling factor attached, that is, it must be of the form *Nx*, where *x* is a character telling what units *N* is in. Common values for *x* are 'i' for inches, 'c' for centimeters, and 'n' for 'ens,' the width of a single character. For example, to indent each section one-half inch, type:

```
.nr si 0.5i
```

The request indents sections by one-half inch per level of depth in the section number. As another example, consider:

```
.nr si 3n
```

which gives three spaces of indent per section depth.

You can produce section headers without automatically generated numbers using:

```
.uh "Title"
```

which will do a section heading, but will not put a number on the section.

## Section Heading Reference

`.sh` *+N T a b c d e f*

Begin numbered section of depth *N*. If *N* is missing, the current depth (maintained in the number register `\n($0)` is used. The values of the individual parts of the section number are maintained in `\n($1` through `\n($6`. There is a `\n(ss` [1v] space before the section. *T* is printed as a section title in font `\n(sf` [8] and size `\n(sp` [10p]. The 'name' of the section may be accessed via `\*($n`. If `\n(si` is non-zero, the base indent is set to `\n(si` times the section depth, and the section title is exdented (see `.ba` in "Miscellaneous Requests"). Also, an additional indent of `\n(so` [0] is added to the section title but not to the body of the section. The font is then set to the paragraph font, so that more information may occur on the line with the section number and title. A `.sh` insures that there is enough room to print the section head plus the beginning of a paragraph, which is about 3 lines total. If you specify *a* through *f*, the section number is set to that number rather than incremented automatically. If any of *a* through *f* are a hyphen that number is not reset. If *T* is a single underscore (_), the section depth and numbering is reset, but the base indent is not reset and nothing is printed. This is useful to automatically coordinate section numbers with chapter numbers.

`.sx` +*N* Go to section depth '*N* [ −1 ]', but do not print the number and title, and do not increment the section number at level *N*. This has the effect of starting a new paragraph at level *N* .

`.uh` *T* Unnumbered section heading. The title *T* is printed with the same rules for spacing, font, etc., as for `.sh`.

`.$p` *T B N*

Print section heading. May be redefined to get fancier headings. *T* is the title passed on the `.sh` or `.uh` line; *B* is the section number for this section, and *N* is the depth of this section. These parameters are not always present; in particular, `.sh` passes all three, `.uh` passes only the first, and `.sx` passes three, but the first two are null strings. Be careful if you redefine this macro, as it is quite complex and subtle.

`.$0` *T B N*

Called automatically after every call to `.$p`. It is normally undefined, but may be used to put every section title automatically into the table of contents, or for some similar function. *T* is the section title for the section title just printed, *B* is the section number, and *N* is the section depth.

`.$1` – `.$6`

Traps called just before printing that depth section. May be defined to give variable spacing before sections. These macros are called from `.$p`, so if you redefine that macro you may lose this feature.

### Parts of the Standard Paper

Some requests help you to format papers. The `.tp` request initializes for a title page. There are no headers or footers on a title page, and unlike other pages, you can space down and leave blank space at the top. For example, source for a typical title page might be:

```
.tp
.sp 2i
.(l C
A BENCHMARK FOR THE NEW SYSTEM
.sp
by
.sp
J. P. Hacker
.)l
.bp
```

The request `.th` sets up the environment of the `nroff` processor to do a thesis. It defines the correct headers, footers, a page number in the upper right-hand corner only, sets the margins correctly, and double spaces.

Use the `.+c` *T* request to start chapters. Each chapter is automatically numbered from one, and a heading is printed at the top of each chapter with the chapter number and the chapter name *T*. For example, to begin a chapter called *Conclusions*, use the request:

```
.+c "CONCLUSIONS"
```

which produces on a new page, the lines

CONCLUSIONS

with appropriate spacing for a thesis. Also, the header is moved to the foot of the page on the first page of a chapter. Although the `.+c` request was not designed to work only with the `.th` request, it is tuned for the format acceptable for a standard PhD thesis.

If the title parameter *T* is omitted from the `.+c` request, the result is a chapter with no heading. You can also use this at the beginning of a paper.

Although papers traditionally have the abstract, table of contents, and so forth at the front, it is more convenient to format and print them last when using `nroff`. This is so that index entries can be collected and then printed for the table of contents. At the end of the paper, give the `.++ P` request, which begins the preliminary part of the paper. After using this request, the `.+c` request will begin a preliminary section of the paper. Most notably, this prints the page number restarted from one in lower case Roman numbers. You may use `.+c` repeatedly to begin different parts of the front material for example, the abstract, the table of contents, acknowledgments, list of illustrations, and so on. You may also use the request `.++ B` to begin the bibliographic section at the end of the paper. For example, the paper might appear as outlined below. (In this figure, comments begin with the sequence `\"`.)

Figure 4-1 *Outline of a Sample Paper*

```
.th                             \" set for thesis mode
.fo ''DRAFT''                   \" define footer for each page
.tp                             \" begin title page
.(l C                           \" center a large block
A BENCHMARK FOR THE NEW SYSTEM
.sp
by
.sp
J.P. Hacker
.)l                             \" end centered part
.+c INTRODUCTION                \" begin chapter named 'INTRODUCTION'
.(x t                           \" make an entry into index 't'
Introduction
.)x                             \" end of index entry
text of chapter one
.+c "NEXT CHAPTER"              \" begin another chapter
.(x t                           \" enter into index 't' again
Next Chapter
.)x
text of chapter two
.+c CONCLUSIONS
.(x t
Conclusions
.)x
text of chapter three
.++ B                           \" begin bibliographic information
.+c BIBLIOGRAPHY                \" begin another 'chapter'
.(x t
Bibliography
.)x
text of bibliography
.++ P                           \" begin preliminary material
.+c "TABLE OF CONTENTS"
.xp t                           \" print index 't' collected above
.+c PREFACE                     \" begin another preliminary section
text of preface
```

## Standard Paper Reference

`.tp` Begin title page. Spacing at the top of the page can occur, and headers and footers are suppressed. Also, the page number is not incremented for this page.

`.th` Set thesis mode. This defines the modes acceptable for a doctoral dissertation. It double spaces, defines the header to be a single page number, and changes the margins to be 1.5 inch on the left and one inch on the top. Use `.++` and `.+c` with it. This macro must be stated before initialization, that is, before the first call of a paragraph macro or `.H`.

`.++` *m H* This request defines the section of the paper you are typing. The section type is defined by *m*: `C` means you are entering the chapter portion of the paper, `A` means you are entering the appendix portion of the paper, `P` means the material following should be the preliminary portion (abstract, table of contents, etc.) of the paper, `AB` means that you are entering the abstract (numbered independently from 1 in Arabic numerals), and `B` means that you are entering the bibliographic portion at the end of the paper. You can also use the variants `RC` and `RA`, which specify renumbering of pages from one at the beginning of each chapter or appendix, respectively. The `H` parameter defines the new header. If there are any spaces in it, the entire header must be quoted. If you want the header to have the chapter number in it, use the string `\\\\n(ch`. For example, to number appendixes 'A.1' etc., type `.++ RA ′′′ \\\\n(ch.%′`. Precede each section (chapter, appendix, etc.) by the `.+c` request. When using `troff`, it is easier to put the front material at the end of the paper, so that the table of contents can be collected and generated; you can then physically move this material to the beginning of the paper.

`.+c` *T* Begin chapter with title *T*. The chapter number is maintained in `\n(ch`. This register is incremented every time `.+c` is called with a parameter. The title and chapter number are printed by `.$c`. The header is moved to the footer on the first page of each chapter. If *T* is omitted, `.$c` is not called; this is useful for doing your own 'title page' at the beginning of papers without a title page proper. `.$c` calls `.$C` as a hook so that chapter titles can be inserted into a table of contents automatically. The footnote numbering is reset to one.

`.$c` *T* Print chapter number (from `\n(ch)` and *T*. You can redefine this macro to your liking. It is defined by default to be acceptable for a standard PhD thesis. This macro calls `$C`, which can be defined to make index entries, or whatever.

`.$C` *K N T*

This macro is called by `.$c`. It is normally undefined, but can be used to automatically insert index entries, or whatever. *K* is a keyword, either 'Chapter' or 'Appendix' (depending on the `.++` mode); *N* is the chapter or appendix number, and *T* is the chapter or appendix title.

`.ac` *A N* This macro (short for `.acm`) sets up the `nroff` environment for photo-ready papers as used by the Association for Computing Machinery (ACM). This format is 25% larger, and has no headers or footers. The author's name *A* is printed at the bottom of the page, but off the part which will be printed in the conference proceedings, together with the current page number and the total number of pages *N*. Additionally, this macro loads the file */usr/lib/me/acm.me*, which may later be augmented with other macros for printing papers for ACM conferences. Note that this macro will not work correctly in `troff`, since it sets the page length wider than the physical width of the phototypesetter roll.

## Two-Column Output — `.2c`

You can get two-column output automatically by using the request `.2c`. This produces everything after it in two-column form. The request `.bc` will start a new column; it differs from `.bp` in that `.bp` may leave a totally blank column when it starts a new page. To revert to single-column output, use `.1c`.

### Column Output Reference

`.2c` *+S N*
: Enter two-column mode. The column separation is set to *+S* [4n, 0.5i in ACM mode] (saved in `\n($s)`. The column width, calculated to fill the single-column line length with both columns, is stored in `\n($l`. The current column is in `\n($c`. You can test register `\n($m` [1] to see if you are in single-column or double-column mode. Actually, the request enters *N*-column [2] output.

`.1c`
: Revert to single-column mode.

`.bc`
: Begin column. This is like `.bp` except that it begins a new column on a new page only if necessary, rather than forcing a whole new page if there is another column left on the current page.

## Defining Macros — `.de`

A *macro* is a collection of requests and text which you may invoke with a simple request. Macros definitions begin with the line `.de` *xx* where *xx* is the name of the macro to be defined, and end with a line consisting of only two dots. After defining the macro, invoking it with the line `.`*xx* is the same as invoking all the other macros. For example, to define a macro that spaces vertically three lines and then centers the next input line, type:

```
.de SS
.sp 3
.ce
..
```

and use it by typing:

```
.SS
Title Line
(beginning of text)
```

Macro names may be one or two characters. In order to avoid conflicts with command names in `-me`, always use upper case letters as names. Avoid the names `TS`, `TH`, `TE`, `EQ`, and `EN`.

## Annotations Inside Keeps

Sometimes you may want to put a footnote or index entry inside a keep. For example, if you want to maintain a 'list of figures', you will want to use something like:

```
.(z
.(c
Text of figure
.)c
.ce
Figure 5.
\!.(x f
\!Figure 5
\!.)x
.)z
```

which will give you a figure with a label and an entry in the index 'f', presumably a list of figures index. Because the index entry is read and interpreted when the keep is read, and not when it is printed, you have is to use the magic string `\!` at the beginning of all the lines dealing with the index. Otherwise, the page number in the index is likely to be wrong. This defers index processing until the figure is generated, and guarantees that the page number in the index is correct. The same comments apply to blocks with `.(b` and `.)b`.

## 4.6. Using `troff` for Phototypesetting

You can prepare documents for either displaying on a workstation or for phototypesetting using the `troff` formatting program.

### Fonts

A *font* is a style of type. There are three fonts that are available simultaneously, Times Roman, Times Italic, and Times Bold, plus the special math font for use with the `eqn` and `neqn` mathematical equation processors. The normal font is Roman. Text which would be underlined in `nroff` with the `.ul` request is set in italics in `troff`.

There are ways of switching between fonts. The requests `.r`, `.i`, and `.b` switch to Roman, italic, and bold fonts respectively. You can set a single word in one of these fonts by typing, for example:

```
.i word
```

which will set *word* in italics but does not affect the surrounding text. In `nroff`, italic and bold text is underlined.

Notice that if you are setting more than one word in a different font, you must surround that word with double quote marks (") so it will appear to the `nroff` processor as a single word. The quote marks will not appear in the formatted text. If you do want a quote mark to appear, quote the entire string even if a single word, and use *two* quote marks where you want one to appear. For example, if you want to produce the text:

> *"Master Control"*

in italics, you must type:

```
.i """Master Control\|"""
```

The \| produces a narrow space so that the '1' does not overlap the quote sign in `troff`.

There are also several *pseudo-fonts* available. For example, the input:

```
.u underlined
```

generates

underlined

and

```
.bx "words in a box"
```

produces

words in a box

You can also get bold italics with

```
.bi "bold italics"
```

Notice that pseudo font requests set only the single parameter in the pseudo font; ordinary font requests will begin setting all text in the special font if you do not provide a parameter. No more than one word should appear with these three font requests in the middle of lines. This is because of the way `troff` justifies text. For example, if you were to give the requests:

```
.bi "some bold italics"
```

and

```
.bx "words in a box"
```

in the middle of a line, `troff` would overwrite the first and the box lines on the second would be poorly drawn.

The second parameter of all font requests is set in the original font. For example, the font request:

```
.b bold face
```

generates 'bold' in bold font, but sets 'face' in the font of the surrounding text, resulting in:

**bold**face

To set the two words 'bold' and 'face' both in **bold face**, type:

```
.b "bold face"
```

You can mix fonts in a word by using the special sequence \c at the end of a line to indicate 'continue text processing'; you can join input lines together without a space between them. For example, the input:

```
.u under \c
.i italics
```

generates unde*italics* , but if you type:

```
.u under
.i italics
```

the result is under *italics* as two words.

## Point Sizes — .sz

The phototypesetter supports different sizes of type, measured in points. The default point size is 10 points for most text and eight points for footnotes. To change the point size, type:

```
.sz +N
```

where *N* is the size wanted in points. The 'vertical spacing,' that is, the distance between the bottom of most letters (the *baseline*) and the adjacent line is set to be proportional to the type size.

Note: Changing point sizes on the phototypesetter is a slow mechanical operation. Consider size changes carefully.

## Fonts and Sizes Reference

`.sz` *+P* The point size is set to *P* [10p], and the line spacing is set proportionally. The ratio of line spacing to point size is stored in **\n($r**. The ratio used internally by displays and annotations is stored in **\n($R**, although `.sz` does not use this.

`.r` *W X* Set *W* in roman font, appending *X* in the previous font. To append different font requests, use '*X* = `\c`. If no parameters, change to roman font.

`.i` *W X* Set *W* in italics, appending *X* in the previous font. If no parameters, change to italic font. Underlines in `nroff`.

`.b` *W X* Set *W* in bold font and append *X* in the previous font. If no parameters, switch to bold font. Underlines in `nroff`.

`.rb` *W X* Set *W* in bold font and append *X* in the previous font. If no parameters, switch to bold font. `.rb` differs from `.b` in that `.rb` does not underline in `nroff`.

`.u` *W X* Underline *W* and append *X*. This is a true underlining, as opposed to the `.ul` request, which changes to 'underline font' (usually italics in `troff`). It won't work right if *W* is spread or broken, which includes being hyphenated, so in other words, it is only safe in nofill mode.

`.q` *W X* Quote *W* and append *X*. In `nroff` this just surrounds *W* with double quote marks (" "), but in `troff` uses directed quotes.

`.bi` *W X* Set *W* in bold italics and append *X*. Actually, sets *W* in italic and overstrikes once. Underlines in `nroff`. It won't work right if *W* is spread or broken, which includes being hyphenated, so it is only safe in nofill mode.

`.bx` *W X* Sets *W* in a box, with *X* appended. Underlines in `nroff`. It won't work right if *W* is spread or broken, which includes being hyphenated, so it is only safe in nofill mode.

### Quotes — `\*(lq` and `\*(rq`

It looks better to use pairs of grave and acute accents to generate double quotes, rather than the double quote character (") on a phototypesetter. For example, compare "quote" to “quote”. In order to make quotes compatible between the typesetter and the workstation or a terminal, use the sequences `\*(lq` and `\*(rq` to stand for the left and right quote respectively. These both appear as " on most terminals, but are typeset as `` and ″ respectively. For example, use:

```
\*(lqSome things aren't true
even if they did happen.\*(rq
```

to generate the result:

> “Some things aren't true even if they did happen.”

As a shorthand, the special font request:

```
.q "quoted text"
```

which generates “quoted text”. Notice that you must surround the material to be quoted with double quote marks if it is more than one word.

## 4.7. Adjusting Macro Parameters

You may adjust a number of macro parameters. You may set fonts to a font number only. In `nroff` font 8 is underlined, and is set in bold font in `troff` (although font 3, bold in `troff`, is not underlined in `nroff`). Font 0 is no font change; the font of the surrounding text is used instead. Notice that fonts 0 and 8 are *pseudo-fonts*; that is, they are simulated by the macros. This means that although it is legal to set a font register to zero or eight, it is not legal to use the escape character form, such as:

```
\f8
```

All distances are in basic units, so it is nearly always necessary to use a scaling factor. For example, the request to set the paragraph indent to eight one-en spaces is:

```
.nr pi 8n
```

and not

```
.nr pi 8
```

which would set the paragraph indent to eight basic units, or about 0.02 inch.

You may use registers and strings of the form `$` *x* in expressions but you should not change them. Macros of the form `$` *x* perform some function as described and may be redefined to change this function. This may be a sensitive operation; look at the body of the original macro before changing it.

On daisy wheel printers in twelve-pitch, you can use the `-rx1` flag to make lines default to one-eighth inch, which is the normal spacing for a newline in twelve-pitch. This is normally too small for easy readability, so the default is to space one-sixth inch.

## 4.8. `roff` Support

`.ix` +*N* Indent, no break. Equivalent to ‘´ in *N*’.

`.bl` *N* Leave *N* contiguous white spaces, on the next page if not enough room on this page. Equivalent to a `.sp` *N* inside a block.

`.pa` +*N* Equivalent to `.bp`.

`.ro` Set page number in Roman numerals. Equivalent to `.af % i`.

`.ar` Set page number in Arabic. Equivalent to `.af % 1`.

`.n1` Number lines in margin from one on each page.

`.n2` *N* Number lines from *N*, stop if N = 0.

`.sk` Leave the next output page blank, except for headers and footers. Use this to leave space for a full-page diagram which is produced externally and pasted in later. To get a partial-page paste-in display, say `.sv` *N*, where *N* is the amount of space to leave; this space will be generated immediately if there is room, and will otherwise be generated at the top of the next page. However, be warned: if *N* is greater than the amount of available space on an *empty* page, no space will be reserved.

## 4.9. Preprocessor Support

`.EQ` *m T* Begin equation. The equation is centered if *m* is `C` or omitted, indented `\n(bi` [4n] if *m* is `I`, and left-justified if *m* is `L`. *T* is a title printed on the right margin next to the equation. See the “Typesetting Mathematics with `eqn`” chapter in this manual for more about equation formatting.

`.EN` *c* End equation. If *c* is `C`, the equation must be continued by immediately following with another `.EQ`, the text of which can be centered along with this one. Otherwise, the equation is printed, always on one page, with `\n(es` [0.5v in `troff`, 1v in `nroff`] space above and below it.

`.TS` *h* Table start. Tables are single-spaced and kept on one page, if possible. If you have a large table that will not fit on one page, use *h* = `H` and follow the header part to be printed on every page of the table with a `.TH`. See the “Formatting Tables with `tbl`” chapter in this manual for more information on laying out tables.

`.TH` With `.TS H`, ends the header portion of the table.

`.TE` Table end. Note that this table does not float, in fact, it is not even guaranteed to stay on one page if you use requests such as `.sp` intermixed with the text of the table. If you want it to float (or if you use requests inside the table), surround the entire table (including the `.TS` and `.TE` requests) with `.(z` and `.)z`.

## 4.10. Predefined Strings

`\**` Footnote number, actually `\*[\n($f\*]`. This macro is incremented after each call to `.)f`.

`\*#` Delayed text number. Actually `[\n($d]`.

`\*[` Superscript. This string gives upward movement and a change to a smaller point size if possible, otherwise it gives the left bracket character ( `[` ). Extra space is left above the line to allow room for the superscript. For example, to produce a superscript you can type `x\*[2\*]`, which will produce $\mathbf{x}^{\mathbf{2}}$.

`\*]` Unsuperscript. Inverse of `\*[`.

`\*<` Subscript. Defaults to < if half-carriage motion not possible. Extra space is left below the line to allow for the subscript.

`\*>` Inverse of `\*<`.

`\*(dw` The day of the week, as a word.

`\*(mo` The month, as a word.

`\*(td` Today's date, directly printable. The date is of the form September 16, 1983. Other forms of the date can be used by using `\n(dy` (the day of the month; for example, 16), `\*(mo` (as noted above) or `\n(mo` (the same, but as an ordinal number; for example, September is 9), and `\n(yr` (the last two digits of the current year).

`\*(lq` Left quote marks; double quote in `nroff`.

`\*(rq` Right quote marks; double quote in `nroff`.

`\*-` An em-dash in `troff`; two hyphens in `nroff`.

## 4.11. Miscellaneous Requests

`.re` Reset tabs. Set to every 0.5i in `troff` and every 0.8i in `nroff`.

`.ba` *+N* Set the base indent to +N [0] (saved in `\n($i`). All paragraphs, sections, and displays come out indented by this amount. Titles and footnotes are unaffected. The `.H` request performs a `.ba` request if `\n(si` [0] is not zero, and sets the base indent to `\n(si*\n($0`.

`.xl` *+N* Set the line length to *N* [6.0i]. This differs from `.ll` because it only affects the current environment.

`.ll` *+N* Set line length in all environments to *N* [6.0i]. Do not use this after output has begun, and particularly not in two-column output. The current line length is stored in `\n($l`.

`.hl` Draws a horizontal line the length of the page. This is useful inside floating keeps to differentiate between the text and the figure.

`.lo` This macro loads another set of macros in `/usr/lib/me/local.me`, which is a set of locally-defined macros. These macros should all be of the form `.*` *X*, where *X* is any letter (upper or lower case) or digit.

## 4.12. Special Characters and Diacritical Marks — .sc

There are a number of special characters and diacritical marks, such as accents, available with -me. To use these characters, you must call the macro .sc to define the characters before using them.

.sc Define special characters and diacritical marks. You must state this macro before initialization.

The special characters available are listed below.

Table 4-1 *Special Characters and Diacritical Marks*

| Name | Usage | Example |
|---|---|---|
| Acute accent | *´ | a*´ ´a |
| Grave accent | *\` | e\*\` \`e |
| Umlaut | *: | u*: ¨u |
| Tilde | *~ | n*~ ~n |
| Caret | *^/e*ˆe̸ | |
| Cedilla | *,/c*,¢ | |
| Czech | *v/e*v/e | |
| Circle | *o | A*o Å |

## 4.13. -me Request Summary

Table 4-2 -me *Request Summary*

| *Request* | *Initial Value* | *Cause Break* | *Explanation* |
|---|---|---|---|
| .(c | – | yes | Begin centered block. |
| .(d | – | no | Begin delayed text. |
| .(f | – | no | Begin footnote. |
| .(l | – | yes | Begin list. |
| .(q | – | yes | Begin major quote. |
| .(x *x* | – | no | Begin indexed item in index *x*. |
| .(z | – | no | Begin floating keep. |
| .)c | – | yes | End centered block. |
| .)d | – | yes | End delayed text. |
| .)f | – | yes | End footnote. |
| .)l | – | yes | End list. |
| .)q | – | yes | End major quote. |
| .)x | – | yes | End index item. |
| .)z | – | yes | End floating keep. |
| .++ *m H* | – | no | Define paper section. *m* defines the part of the paper and can be C (chapter), A (appendix), P (preliminary, for example, abstract, table of contents, etc.), B (bibliography), RC (chapters renumbered from page one each chapter), or RA (appendix renumbered from page one). |

Table 4-2 -me *Request Summary— Continued*

| *Request* | *Initial Value* | *Cause Break* | *Explanation* |
|---|---|---|---|
| .+c *T* | – | yes | Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title. |
| .1c | 1 | yes | One-column format on a new page. |
| .2c | 1 | yes | Two-column format. |
| .EN | – | yes | Space after equation produced by eqn or neqn. |
| .EQ *x y* | – | yes | Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be I to indent equation (default), L to left-adjust the equation, orC to center the equation. |
| .TE | – | yes | End table. |
| .TH | – | yes | End heading section of table. |
| .TS *x* | – | yes | Begin table; if *x* is H, table has repeated heading. |
| .ac *A N* | – | no | Set up for ACM-style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization. |
| .b *x* | no | yes | Print *x* in boldface; if no argument switch to boldface. |
| .ba +*n* | 0 | yes | Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs). |
| .bc | no | yes | Begin new column. |
| .bi *x* | no | no | Print *x* in bold italics (nofill only). |
| .bx *x* | no | no | Print *x* in a box (nofill only). |
| .ef '*x*'*y*'*z*' | '''' | no | Set even footer to *x y z*. |
| .eh '*x*'*y*'*z*' | '''' | no | Set even header to *x y z*. |
| .fo '*x*'*y*'*z*' | '''' | no | Set footer to *x y z*. |
| .he '*x*'*y*'*z*' | '''' | no | Set header to *x y z*. |
| .hl | – | yes | Draw a horizontal line. |
| .hx | – | no | Suppress headers and footers on next page. |
| .i *x* | no | no | Italicize *x*; if *x* is missing, italic text follows. |
| .ip *x y* | no | yes | Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5). |
| .lp | yes | yes | Start left-block paragraph. |
| .lo | – | no | Read in a file of local macros of the form .**x*. Must be given before initialization. |
| .np | 1 | yes | Start numbered paragraph. |
| .of '*x*'*y*'*z*' | '''' | no | Set odd footer to *x y z*. |
| .oh '*x*'*y*'*z*' | '''' | no | Set odd header to *x y z*. |
| .pd | – | yes | Print delayed text. |
| .pp | no | yes | Begin paragraph. First line indented. |
| .r | yes | no | Roman text follows. |
| .re | – | no | Reset tabs to default values. |
| .sc | – | no | Read in a file of special characters and diacritical marks. Must be given before initialization. |

Table 4-2 –me *Request Summary— Continued*

| ***Request*** | ***Initial Value*** | ***Cause Break*** | ***Explanation*** |
|---|---|---|---|
| `.sh` *n x* | – | yes | Section head follows, font automatically bold. *n* is level of section, *x* is title of section. |
| `.sk` | no | no | Leave the next page blank. Only one page is remembered ahead. |
| `.sz` +*n* | 10p | no | Increase the point size by *n* points. |
| `.th` | no | no | Produce the paper in thesis format. Must be given before initialization. |
| `.tp` | no | yes | Begin title page. |
| `.u` *x* | – | no | Underline argument (even in `troff`) (nofill only). |
| `.uh` | – | yes | Like '.sh' but unnumbered. |
| `.xp` *x* | – | no | Print index *x*. |

5

# refer — A Bibliography System

# 5

# refer — A Bibliography System

## 5.1. Introduction

refer is a bibliography system that supports data entry, indexing, retrieval, sorting, runoff, convenient citations, and footnote or endnote numbering. You can enter new bibliographic data into the database, index the selected data, and retrieve bibliographic references from the database. This document assumes you know how to use a Unix editor, and that you are familiar with the nroff and troff text formatters.

The refer program is a preprocessor for nroff and troff, and works like like eqn and tbl. refer is used for literature citations, rather than for equations and tables. Given incomplete but sufficiently precise citations, refer finds references in a bibliographic database. The complete references are formatted as footnotes, numbered, and placed either at the bottom of the page, or at the end of a chapter.

A number of related programs make refer easier to use. The addbib program is for creating and extending the bibliographic database; sortbib sorts the bibliography by author and date, or other selected criteria; and roffbib runs off the entire database, formatting it not as footnotes, but as a bibliography or annotated bibliography.

Once a full bibliography has been created, access time can be improved by making an index to the references with indxbib. Then, the lookbib program can be used to quickly retrieve individual citations or groups of citations. Creating this inverted index will speed up refer, and lookbib will allow you to verify that a citation is sufficiently precise to deliver just one reference.

## 5.2. Features

Taken together, the refer programs constitute a database system for use with variable-length information. To distinguish various types of bibliographic material, the system uses *labels* composed of upper case letters, preceded by a percent sign and followed by a space. For example, one document might be given this entry:

```
%A Joel Kies
%T Document Formatting on Unix Using the -ms Macros
%I Computing Services
%C Berkeley
%D 1980
```

Each line is called a *field*, and lines grouped together are called a *record*; records are separated from each other by a blank line. Bibliographic information follows the labels. This field contains *data* to be used by the `refer` system. The order of fields is not important, except that authors should be entered in the same order as they are listed on the document. Fields can be as long as necessary, and may even be continued on the following line(s).

The labels are meaningful to `nroff` and `troff` macros, and, with a few exceptions, the `refer` program itself does not pay attention to the labels. This implies that you can change the label codes, if you also change the macros used by `nroff` and `troff`. The macro package takes care of details like proper ordering, underlining the book title or journal name, and quoting the article's title. Here are the labels used by `refer`, with an indication of what they represent:

| | |
|---|---|
| %H | Header commentary, printed before reference |
| %A | Author's name |
| %Q | Corporate or foreign author (unreversed) |
| %T | Title of article or book |
| %S | Series title |
| %J | Journal containing article |
| %B | Book containing article |
| %R | Report, paper, or thesis (for unpublished material) |
| %V | Volume |
| %N | Number within volume |
| %E | Editor of book containing article |
| %P | Page number(s) |
| %I | Issuer (publisher) |
| %C | City where published |
| %D | Date of publication |
| %O | Other commentary, printed at end of reference |
| %K | Keywords used to locate reference |
| %L | Label used by –k option of `refer` |
| %X | Abstract (used by `roffbib`, not by `refer`) |

Only relevant fields (lines) should be supplied. Except for %A, the author field, each field should be given only once. In the case of multiple authors, the senior author should be entered first. Your entry in such a case, might look like this:

```
%A Brian W. Kernighan
%A P. J. Plauger
%T Software Tools in Pascal
%I Addison-Wesley
%C Reading, Massachusetts
%D 1981
```

The %Q is for organizational authors, or authors with Japanese or Arabic names, in which cases there is no clear last name. Books should be labeled with the %T, not with the %B, which is reserved for books containing articles. The %J and %B fields should never appear together, although if they do, the %J will override the %B. If there is no author, just an editor, it is best to type the editor in the %A field, as in this example:

```
%A Bertrand Bronson, ed.
```

The %E field is used for the editor of a book (%B) containing an article, which has its own author. For unpublished material such as theses, use the %R field; the title in the %T field will be quoted, but the contents of the %R field will not be underlined. Unlike other fields, %H, %O, and %X should contain their own punctuation. Here is an example:

```
%A Mike E. Lesk
%T Some Applications of Inverted Indexes on the Unix System
%B Unix Programmer's Manual
%I Bell Laboratories
%C Murray Hill, NJ
%D 1978
%V 2a
%K refer mkey inv hunt
%X Difficult to read paper that dwells on indexing strategies,
giving little practical advice about using \fBrefer\fP.
```

Note that the author's name is given in normal order, without inverting the surname; inversion is done automatically, except when %Q is used instead of %A. We use %X rather than %O for the commentary because we do not want the comment printed every time the reference is used. The %O and %H fields are printed by both `refer` and `roffbib;` the %X field is printed only by `roffbib,` as a detached annotation paragraph.

## 5.3. Data Entry with `addbib`

The `addbib` program is for creating and extending bibliographic databases. You must give it the filename of your bibliography:

```
hostname% addbib database
```

Every time you enter `addbib`, it asks if you want instructions. To get them, type **y**; to skip them, type RETURN. `addbib` prompts for various fields, reads from the keyboard, and writes records containing the `refer` codes to the database. After finishing a field entry, you should end it by typing RETURN. If a field is too long to fit on a line, type a backslash (\) at the end of the line, and you will be able to continue on the following line. Note: the backslash works in this capacity only inside `addbib`.

A field will not be written to the database if nothing is entered into it. Typing a minus sign as the first character of any field will cause `addbib` to back up one field at a time. Backing up is the best way to add multiple authors, and it really helps if you forget to add something important. Fields not contained in the prompting skeleton may be entered by typing a backslash as the last character before RETURN. The following line will be sent verbatim to the database and `addbib` will resume with the next field. This is identical to the procedure for dealing with long fields, but with new fields, don't forget the % key-letter.

Finally, you will be asked for an abstract (or annotation), which will be preserved as the %X field. Type in as many lines as you need, and end with a control-D (hold down the CTRL button, then press the "d" key). This prompting for an abstract can be suppressed with the –a command line option.

After one bibliographic record has been completed, addbib will ask if you want to continue. If you do, type RETURN ; to quit, type **q** or **n** (quit or no). It is also possible to use one of the system editors to correct mistakes made while entering data. After the Continue? prompt, type any of the following: **edit**, **ex**, **vi**, or **ed** — you will be placed inside the corresponding editor, and returned to addbib afterwards, from where you can either quit or add more data.

If the prompts normally supplied by addbib are not enough, are in the wrong order, or are too numerous, you can redefine the skeleton by constructing a promptfile. Create some file, to be named after the –p command line option. Place the prompts you want on the left side, followed by a single TAB (control-I), then the refer code that is to appear in the bibliographic database. addbib will send the left side to the screen, and the right side, along with data entered, to the database.

## 5.4. Printing the Bibliography

sortbib is for sorting the bibliography by author (%A) and date (%D), or by data in other fields. Sortbib is quite useful for producing bibliographies and annotated bibliographies, which are seldom entered in strict alphabetical order.

Sortbib takes as arguments the names of up to 16 bibliography files, and sends the sorted records to standard output (the terminal screen), which may be redirected through a pipe or into a file.

The -s*KEYS* flag to sortbib will sort by fields whose key-letters are in the *KEYS* string, rather than merely by author and date. Key-letters in *KEYS* may be followed by a + to indicate that all such fields are to be used. The default is to sort by senior author and date (printing the senior author last name first), but –sA+D will sort by all authors and then date, and –sATD will sort on senior author, then title, and then date.

roffbib is for running off the (probably sorted) bibliography. It can handle annotated bibliographies — annotations are entered in the %X (abstract) field. roffbib is a shell script that calls refer –B and nroff –mbib. It uses the macro definitions that reside in /usr/lib/tmac/tmac.bib, which you can redefine if you know nroff and troff. Note that refer will print the %H and %O commentaries, but will ignore abstracts in the %X field; roffbib will print both fields, unless annotations are suppressed with the –x option.

The following command sequence will lineprint the entire bibliography, organized alphabetically by author and date:

```
hostname% sortbib database | roffbib | lpr
```

This is a good way to proofread the bibliography, or to produce a stand-alone bibliography at the end of a paper. Incidentally, roffbib accepts all flags used with nroff. For example:

```
hostname% sortbib database | roffbib -Txerox -s1
```

will make accent marks work on a Xerox printer, and stop at the bottom of every page for changing paper. The –n and –o flags may also be quite useful, to start page numbering at a selected point, or to produce only specific pages.

roffbib understands four command-line number registers: N, V, L, and O. These are something like the two-letter number registers in **-ms**. The –rN1 argument will number references beginning at one (1); use another number to start somewhere besides one. The –rV2 flag will double-space the entire bibliography, while –rV1 will double-space the references, but single-space the annotation paragraphs. Finally, specifying –rL6i changes the line length from 6.5 inches to 6 inches, and saying –rO1i sets the page offset to one inch, instead of zero. (That's a capital O after –r, not a zero.)

## 5.5. Citing Papers with refer

The refer program normally copies input to output, except when it encounters an item of the form:

```
.[
partial citation
.]
```

The partial citation may be just an author's name and a date, or perhaps a title and a keyword, or maybe just a document number. refer looks up the citation in the bibliographic database, and transforms it into a full, properly-formatted reference. If the partial citation does not correctly identify a single work (either finding nothing, or more than one reference), a diagnostic message is given. If nothing is found, it will say "No such paper." If more than one reference is found, it will say "Too many hits." Other diagnostic messages can be quite cryptic; if you are in doubt, use checknr to verify that all your .[ s have matching .] s.

When everything goes well, the reference will be brought in from the database, numbered, and placed at the bottom of the page. This citation, for example, was produced by:

```
This citation,
.[
lesk inverted indexes
.]
for example, was produced by
```

The .[ and .] markers, in essence, replace the .FS and .FE of the **-ms** macros, and also provide a numbering mechanism. Footnote numbers will be bracketed

[1] Mike E. Lesk, "Some Applications of Inverted Indexes on the Unix System," in *Unix Programmer's Manual*, Bell Laboratories, Murray Hill, NJ, 1978.

on the lineprinter, but superscripted on daisy-wheel terminals and in `troff`. In the reference itself, articles will be quoted, and books and journals will be underlined in `nroff`, and italicized in `troff`.

Sometimes you need to cite a specific page number along with more general bibliographic material. You may have, for instance, a single document that you refer to several times, each time giving a different page citation. This is how you could get "p. 10" in the reference:

```
.[
kies document formatting
%P 10
.]
```

The first line, a partial citation, will find the reference in your bibliography. The second line will insert the page number into the final citation. Ranges of pages may be specified as "%P 56-78".

When the time comes to run off a paper, you will need to have two files: the bibliographic database, and the paper to format. Use a command line something like one of these:

```
hostname% refer -p database paper | nroff -ms
hostname% refer -p database paper | tbl | nroff -ms
hostname% refer -p database paper | tbl | neqn | nroff -ms
```

If other preprocessors are used, `refer` should precede `tbl`, which must in turn precede `eqn`, or `neqn`. The `-p` option specifies a "private" database, which most bibliographies are.

### 5.6. `refer` Command Line Options

Many people like to place references at the end of a chapter, rather than at the bottom of the page. The `-e` option will accumulate references until a macro sequence of the form

```
.[
$LIST$
.]
```

is encountered (or until the end of file). `refer` will then write out all references collected up to that point, collapsing identical references. Warning: there is a limit (currently 200) on the number of references that can be accumulated at one time.

It is also possible to sort references that appear at the end of text. The `-s`*KEYS* flag will sort references by fields whose key-letters are in the *KEYS* string, and permute reference numbers in the text accordingly. It is unnecessary to use `-e` with the `-s`*KEYS* flag, since `-s` implies `-e`. See the section "Printing the Bibliography" for additional features of the `-s`*KEYS* flag.

`refer` can also make citations in what is known as the Social or Natural Sciences format. Instead of numbering references, the `-l` (letter ell) flag makes labels from the senior author's last name and the year of publication. For example, a reference to the paper on Inverted Indexes cited above might appear as [Lesk1978a]. It is possible to control the number of characters in the last name, and the number of digits in the date. For instance, the command line argument −l6,2 might produce a reference such as [Kernig78c].

Some bibliography standards shun both footnote numbers and labels composed of author and date, requiring some keyword to identify the reference. The −k flag indicates that, instead of numbering references, key labels specified on the %L line should be used to mark references.

The −n flag means to not search the default reference file, located in /usr/dict/papers/Rv7man. Using this flag may make refer marginally faster. The −a*n* flag will reverse the first *n* author names, printing Jones, J. A. instead of J. A. Jones. Often −a1 is enough; this will reverse the first and last names of only the senior author. In some versions of refer there is also the −f flag to set the footnote number to some predetermined value; for example, −f23 would start numbering with footnote 23.

## 5.7. Making an Index

Once your database is large and relatively stable, it is a good idea to make an index to it, so that references can be found quickly and efficiently. The indxbib program makes an inverted index to the bibliographic database (this program is called pubindex in the Bell Labs manual). An inverted index could be compared to the thumb cuts of a dictionary — instead of going all the way through your bibliography, programs can move to the exact location where a citation is found.

indxbib itself takes a while to run, and you will need sufficient disk space to store the indexes. But once it has been run, access time will improve dramatically. Furthermore, large databases of several million characters can be indexed with no problem. The program is exceedingly simple to use:

```
hostname% indxbib database
```

Be aware that changing your database will require that you run indxbib over again. If you don't, you may fail to find a reference that really is in the database.

Once you have built an inverted index, you can use lookbib to find references in the database. lookbib cannot be used until you have run indxbib. When editing a paper, lookbib is very useful to make sure that a citation can be found as specified. It takes one argument, the name of the bibliography, and then reads partial citations from the terminal, returning references that match, or nothing if none match. Its prompt is the greater-than sign.

```
hostname% lookbib database
Instructions? n
> lesk inverted indexes
%A Mike E. Lesk
%T Some Applications of Inverted Indexes on the Unix System
%J Unix Programmer's Manual
%I Bell Laboratories
%C Murray Hill, NJ
%D 1978
%V 2a
%X Difficult to read paper that dwells on indexing strategies,
giving little practical advice about using \fLrefer\fP.
>
```

If more than one reference comes back, you will have to give a more precise citation for `refer`. Experiment until you find something that works; remember that it is harmless to overspecify.

To get out of the `lookbib` program, type a CTRL-D alone on a line; `lookbib` then exits with an "`EOT`" message.

`lookbib` can also be used to extract groups of related citations. For example, to find all the papers by Brian Kernighan in the system database, and send the output to a file, type:

```
hostname% lookbib /usr/dict/papers/Ind > kern.refs
Instructions ? n
> kernighan
> CTRL-D
EOT
hostname% cat kern.refs
```

Your file, "kern.refs", will be full of references. A similar procedure can be used to pull out all papers of some date, all papers from a given journal, all papers containing a certain group of keywords, etc.

## 5.8. `refer` Bugs and Some Solutions

### Blanks at Ends of Lines

The `refer` program will mess up if there are blanks at the end of lines, especially the %A author line. `addbib` carefully removes trailing blanks, but they may creep in again during editing. Use an `ex` editor command —

```
g/ *$/s///
```

— or similar method to remove trailing blanks from your bibliography.

### Interpolated Strings

Having bibliographic fields passed through as string definitions implies that interpolated strings (such as accent marks) must have two backslashes, so they can pass through copy mode intact. For instance, the word "téléphone" would have to be represented:

```
te\\*´le\\*´phone
```

in order to come out correctly. In the %X field, by contrast, you will have to use single backslashes instead. This is because the %X field is not passed through as a string, but as the body of a paragraph macro.

### Interpreting Foreign Surnames

Another problem arises from authors with foreign names. When a name like "Valéry Giscard d'Estaing" is turned around by the –a option of refer, it will appear as "d'Estaing, Valéry Giscard," rather than as "Giscard d'Estaing, Valéry." To prevent this, enter names as follows:

```
%A Vale\\*´ry Giscard\0d'Estaing
%A Alexander Csoma\0de\0Ko\\*:ro\\*:s
```

(The second is the name of a famous Hungarian linguist.) The backslash-zero is an nroff and troff request meaning to insert a digit-width space. Because the seccnd argument to the %A field contains no blank spaces to confuse the refer program, refer will treat the second field as a single word. This protects against faulty name reversal, and also against mis-sorting.

### Footnote Numbers

Footnote numbers are placed at the end of the line before the .[ macro. This line should be a line of text, not a macro. As an example, if the line before the .[ is a .R macro, then the .R will eat the footnote number. (The .R is an -ms request meaning change to Roman font.) In cases where the font needs changing, it is necessary to use the following method immediately before the citation:

```
Aho \fIet al.\fP
.[
awk aho kernighan weinberger
.]
```

Now the reference will be to Aho *et al.*[2] The \fI changes to italics, and the \fR changes back to Roman font. Both these requests are nroff and troff requests, not part of -ms. If and when a footnote number is added after this sequence, it will indeed appear in the output.

[2] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *Awk — A Pattern Scanning and Text Processing Language*, Bell Laboratories, Murray Hill, NJ.

## 5.9. Internal Details of `refer`

You have already read everything you need to know in order to use the `refer` bibliography system. The remaining sections are provided only for extra information, and in case you need to change the way `refer` works.

The output of `refer` is a stream of string definitions, one for each field in a reference. To create string names, percent signs are simply changed to an open bracket, and an [F string is added, containing the footnote number. The %X, %Y and %Z fields are ignored; however, the `annobib` program changes the %X to an `.AP` (annotation paragraph) macro. The Lesk citation used above yields this intermediate output:

```
.ds [F 1
.]-
.ds [A Mike E. Lesk
.ds [T Some Applications of Inverted Indexes on the Unix System
.ds [J Unix Programmer's Manual
.ds [I Bell Laboratories
.ds [C Murray Hill, NJ
.ds [D 1978
.ds [V 2a
.nr [T 0
.nr [A 0
.nr [O 0
.][ 1 journal-article
```

These string definitions are sent to `nroff`, which can use the **-ms** macros defined in `/usr/lib/mx/ms.xref` to take care of formatting things properly. The initializing macro `.]-` precedes the string definitions, and the labeled macro `.][` follows. These are changed from the input `.[` and `.]` so that running a file twice through `refer` is harmless.

The `.][` macro, used to print the reference, is given a type-number argument, which is a numeric label indicating the type of reference involved. Here is a list of the various kinds of references:

| Field | Value | Kind of Reference |
|---|---|---|
| %J | 1 | Journal Article |
| %B | 3 | Article in Book |
| %G | 4 | Report, Government Report |
| %I | 2 | Book |
| %M | 5 | Bell Labs Memorandum (undefined) |
| none | 0 | Other |

The order listed above is indicative of the precedence of the various fields. In other words, a reference that has both the %J and %B fields will be classified as a journal article. If none of the fields listed is present, then the reference will be classified as "other."

The footnote number is flagged in the text with the following sequence, where *number* is the footnote number:

```
\*([.number\*(.]
```

The *([. and *(.] stand for bracketing or superscripting. In `nroff` with low-resolution devices such as the lpr and a crt, footnote numbers will be bracketed. In `troff,` or on daisy-wheel printers, footnote numbers will be superscripted. Punctuation normally comes before the reference number; this can be changed by using the `-P` (postpunctuation) option of `refer`.

In some cases, it is necessary to override certain fields in a reference. For instance, each time a work is cited, you may want to specify different page numbers, and you may want to change certain fields. This citation will find the Lesk reference, but will add specific page numbers to the output, even though no page numbers appeared in the original reference.

```
.[
lesk inverted indexes
%P 7-13
%I Computing Services
%O UNX 12.2.2.
.]
```

The %I line will also override any previous publisher information, and the %O line will append some commentary. The `refer` program simply adds the new %P, %I, and %O strings to the output, and later strings definitions cancel earlier ones.

It is also possible to insert an entire citation that does not appear in the bibliographic database. This reference, for example, could be added as follows:

```
.[
%A Brian Kernighan
%T A troff Tutorial
%I Bell Laboratories
%D 1978
.]
```

This will cause `refer` to interpret the fields exactly as given, without searching the bibliographic database. This practice is not recommended, however, because it's better to add new references to the database, so they can be used again later.

If you want to change the way footnote numbers are printed, signals can be given on the `.[` and `.]` lines. For example, to say "See reference (2)," the citation should appear as:

```
See reference
.[(
partial citation
.]),
```

Note that blanks are significant on these signal lines. If a permanent change in the footnote format is desired, it is best to redefine the [. and .] strings.

## 5.10. Changing the refer Macros

This section is provided for those who wish to rewrite or modify the refer macros. This is necessary in order to make output correspond to specific journal requirements, or departmental standards. First there is an explanation of how new macros can be substituted for the old ones. Then several alterations are given as examples.

The refer macros for nroff and troff supplied by the **-ms** macro package reside in /usr/lib/ms/ms.xref; they are reference macros, for producing footnotes or endnotes. The refer macros used by roffbib, on the other hand, reside in /usr/lib/tmac/tmac.bib; they are for producing a stand-alone bibliography.

To change the macros used by roffbib, you will need to get your own version of this shell script into the directory where you are working. This command will get you a copy of roffbib and the macros it uses:

```
hostname% cp /usr/lib/tmac/tmac.bib bibmac
```

You can proceed to change bibmac as much as you like. Then when you use roffbib, you should specify your own version of the macros, which will be substituted for the normal ones

```
hostname% roffbib -m bibmac filename
```

where *filename* is the name of your bibliography file. Make sure there's a space between –m and bibmac.

If you want to modify the refer macros for use with nroff and the **-ms** macros, you will need to get a copy of "ms.ref":

```
hostname% cp /usr/lib/ms/ms.ref refmac
```

These macros are much like "bibmac", except they have .FS and .FE requests, to be used in conjunction with the **-ms** macros, rather than independently defined .XP and .AP requests. Now you can put this line at the top of the paper to be formatted:

```
.so refmac
```

Your new refer macros will override the definitions previously read in by the **-ms** package. This method works only if "refmac" is in the working directory.

Suppose you didn't like the way dates are printed, and wanted them to be parenthesized, with no comma before. There are five identical lines you will have to change. The first line below is the old way, while the second is the new way:

```
.if !"\\*([D"" , \\*([D\c
.if !"\\*([D"" \& (\\*([D)\c
```

In the first line, there is a comma and a space, but no parentheses. The "\c" at the end of each line indicates to `nroff` that it should continue, leaving no extra space in the output. The "\&" in the second line is the do-nothing character; when followed by a space, a space is sent to the output.

If you need to format a reference in the style favored by the Modern Language Association or Chicago University Press, in the form (city: publisher, date), then you will have to change the middle of the book macro [2 as follows:

```
\& (\c
.if !"\\*([C"" \\*([C:
\\*([I\c
.if !"\\*([D"" , \\*([D\c
)\c
```

This would print (Berkeley: Computing Services, 1982) if all three strings were present. The first line prints a space and a parenthesis; the second prints the city (and a colon) if present; the third always prints the publisher (books must have a publisher, or else they're classified as other); the fourth line prints a comma and the date if present; and the fifth line closes the parentheses. You would need to make similar changes to the other macros as well.

# 6

# Formatting Tables with tbl

# 6

# Formatting Tables with tbl

This chapter provides instructions for preparing tbl input to format tables and for running the tbl preprocessor on a file.[1] It also supplies numerous examples after which to pattern your own tables. The description of instructions is precise but technical, and the newcomer may prefer to glance over the examples first, as

tbl turns a simple description of a table into a troff or nroff program that prints the table. From now on, unless noted specifically, we'll refer to both troff and nroff as troff since tbl treats them the same. tbl makes phototypesetting tabular material relatively simple compared to normal typesetting methods. You may use tbl with the equation formatting program eqn or various layout macro packages, as tbl does not duplicate their functions.

Tables are made up of columns which may be independently centered, right-adjusted, left-adjusted, or aligned by decimal points. Headings may be placed over single columns or groups of columns. A table entry may contain equations, or may consist of several rows of text. Horizontal or vertical lines may be drawn as desired in the table, and any table or element may be enclosed in a box. For example:

| 1970 Federal Budget Transfers<br>(in billions of dollars) | | | |
|---|---|---|---|
| State | Taxes<br>collected | Money<br>spent | Net |
| New York | 22.91 | 21.35 | –1.56 |
| New Jersey | 8.33 | 6.96 | –1.37 |
| Connecticut | 4.12 | 3.10 | –1.02 |
| Maine | 0.74 | 0.67 | –0.07 |
| California | 22.29 | 22.42 | +0.13 |
| New Mexico | 0.70 | 1.49 | +0.79 |
| Georgia | 3.30 | 4.28 | +0.98 |
| Mississippi | 1.15 | 2.32 | +1.17 |
| Texas | 9.33 | 11.13 | +1.80 |

[1] The material in this chapter is derived from *Tbl — A Program to Format Tables*, M.E. Lesk, Bell Laboratories, Murray Hill, New Jersey.

The input to `tbl` is text for a document, with the text preceded by a `.TS` (table start) command and followed by a `.TE` (table end) command. `tbl` processes the tables, generating `troff` formatting commands, and leaves the remainder of the text unchanged. The `.TS` and `.TE` lines are copied, too, so that `troff` page layout macros, such as the formatting 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 as follows:

```
. . .
ordinary text of your document
. . .
.TS
first table
.TE
. . .
ordinary text of your document
. . .
.TS
second table
.TE
. . .
ordinary text of your document
. . .
```

where the format of each table is as follows:

```
.TS
options for the table  ;
format describing the layout of the table  .
data to be laid out in the table
                            .
                            .
                            .
data to be laid out in the table
.TE
```

Each table is independent, and must contain formatting information, indicated by *format describing the layout of the table*, followed by the *data to be laid out in the table*. You may precede the formatting information, which describes the individual columns and rows of the table, by *options for the table* that affect the entire table.

## 6.1. Running tbl

You can run tbl on a simple table by piping the tbl output to troff (or your installation's equivalent for the phototypesetter) with the command:

```
hostname% tbl file | troff -options
```

where *file* is the name of the file you want to format. For more complicated use, where there are several input files, and they contain equations and **-ms** macro package requests as well as tables, the normal command is:

```
hostname% tbl file1 file2... | eqn | troff -ms
```

You can, of course, use the usual options on the troff and eqn commands. The usage for nroff is similar to that for troff, but only printers such as the TELETYPE® Model 37 and Diablo-mechanism (DASI or GSI) or other printers that can handle reverse paper motions can print boxed tables directly. If you are running tbl on a line printer that does not filter reverse paper motions, use the col processor to filter the multicolumn output.

If you are using an IBM 1403 line printer 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 options recognized by tbl are macro package specifications such as **-ms** and **-mm**. These options are turned into commands to fetch the corresponding macro files; usually it is more convenient to place these arguments on the troff part of the command line, tbl accepts them as well.

*Caveats*: Note that when you use eqn and tbl together on the same file, put tbl 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, using the delim mechanism in eqn, you must put tbl first or the output will be scrambled.

Also, beware 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. To avoid this, use the delim(*xx* ) table option to prevent splitting numerical columns within the delimiters.

For example, if the eqn delimiters are $$, giving delim($$) a numerical column such as 1245±16, means the column entry will not be divided after 1245, but after 16. This is the output: '1245±16' (all in one column within the table).

The only recommended in-line equation delimiters inside tables (tbl) are $$ or @@. Most of the other special characters have special meanings either inside eqn or tbl.

Some versions of tbl limit 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. Avoid using troff number registers used by tbl 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 lower-case 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 you may not use them in the same column.

For aid in writing layout 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. More importantly, to assist in laying out multi-page 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. Use of this macro in the page footer boxes a multi-page table. In particular, you can use the **-ms** macros to print a multi-page 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 there aren't any. 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. For example:

```
.TS H
center box tab (/);
c s
l l .
Employees
_
Name/Phone
_
.TH
Jonathan Doe/123-4567
< etc. >
.TE
```

Note that this is *not* a feature of `tbl`, but of the **-ms** layout macros.

## 6.2. Input Commands

As indicated above, a table contains, first, global options, then a format section describing the layout of the table entries, and then the data to be printed. The format and data are always required, but not the options. The sections that follow explain how to enter the various parts of the table.

## Options That Affect the Whole Table

There may be a single line of options affecting the whole table. If present, this line must follow the `.TS` line immediately, 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` center the table (default is left-adjusted).

`expand` make the table as wide as the current line length.

`box` enclose the table in a box.

`allbox` enclose each item in the table in a box.

`doublebox` enclose the table in two boxes — a frame.

`tab(`*x*`)` use *x* instead of tab to separate data items.

`linesize(`*n*`)` set lines or rules (such as from `box`) in *n* point type.

`delim(`*xy*`)` recognize *x* and *y* as the *eqn* delimiters.

A standard option line is:

```
center box tab (/) ;
```

which centers the table on the page, draws a box around it, and uses the slash '/' character as the column separator for data items.

The `tbl` program tries to keep boxed tables on one page by issuing appropriate `troff` 'need' (`.ne`) commands. These requests are calculated from the number of lines in the tables, so if there are spacing commands embedded in the input, these requests may be inaccurate. Use normal `troff` procedures, such as keep-release macros, in this case. If you must have a multi-page boxed table, use macros designed for the purpose, as explained above under *Running 'tbl'*.

## Key Letters — Format Describing Data Items

The format section of the table specifies the layout of the columns. Each line in this section corresponds to one line of the table, except that the last line corresponds to all following lines up to the next `.T&`, if present as shown below. Each line contains a *key-letter* for each column of the table. It is good practice to separate the key letters for each column by spaces, tabs, or a visible character such as a slash '/'. Each key-letter is one of the following:

`L` or `l` indicates a left-adjusted column entry.

`R` or `r` indicates a right-adjusted column entry.

`C` or `c` indicates a centered column entry.

`N` or `n` indicates a numerical column entry, to line up the units digits of numerical entries.

`A` or `a` indicates an alphabetic subcolumn; all corresponding entries are aligned on the left, and positioned so that the widest is centered within the column (see the "Some London Transport Statistics" example).

S or s indicates a spanned heading; that is, it indicates that the entry from the previous column continues across this column; not allowed for the first column.

indicates a vertically spanned heading; that is, it indicates that the entry from the previous row continues down through this row; not allowed for the first row of the table.

When you specify numerical alignment, tbl requires a location for the decimal point. The rightmost dot (.) adjacent to a digit is used as a decimal point; if there is no dot adjoining a digit, the rightmost digit is used as a units digit; if no alignment is indicated, the item is centered in the column. However, you may use the special non-printing character string \& to override unconditionally dots and digits, or to align alphabetic data; this string lines up where a dot normally would, and 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:

```
13              13
4.2              4.2
26.4.12       26.4.12
abc              abc
abc\&           abc
43\&3.22        433.22
749.12          749.12
```

**Note:** If numerical data are used in the same column with wider L or r type table entries, the widest *number* is centered relative to the wider L or r items (we use L here instead of l for readability; they have the same meaning as key-letters). Alignment within the numerical items is preserved. This is similar to the way a type data are formatted, as explained above. However, alphabetic subcolumns (requested by the a key-letter) are always slightly indented relative to L items; if necessary, the column width is increased to force this. This is not true for n type entries.

Note: Do not use the n and a items in the same column.

For readability, separate the key-letters describing each column with spaces. Indicate the end of the format section by a period. The layout of the key-letters in the format section resembles the layout of the actual data in the table. Thus a simple format is:

```
.TS
c  s  s
l  n  n .
text
.TE
```

which specifies a table of three columns. The first line of the table contains a centered heading that spans across all three columns; each remaining line contains a left-adjusted item in the first column followed by two columns of numerical data.

A sample table in this format is:

| | | |
|---|---|---|
| | Overall title | |
| Item-a | 34.22 | 9.1 |
| Item-b | 12.65 | .02 |
| Items: c,d,e | 23 | 5.8 |
| Total | 69.87 | 14.92 |

## Optional Features of Key Letters

There may be extra information following a key-letter that modifies its basic behavior. Additional features of the key-letter system follow:

*Horizontal lines*
— A key-letter may be replaced by '_' (underscore) to indicate a horizontal line in place of the corresponding column entry, or by '=' to indicate a double horizontal line. You can also type this in the data portion. 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 displayed.

*Vertical lines*
— A vertical bar may be placed between column key-letters. This draws a vertical line between the corresponding columns of the table. A vertical bar to the left of the first key-letter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between key-letters, a double vertical line is drawn.

*Space between columns*
— A number may follow the key-letter. 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')[2]. If the `expand` option is used, these numbers are multiplied by a constant such that the table is as wide as the current line length. The default column separation number is 3. If the separation is changed, the 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 key-letter is followed by `t` or `T`, any corresponding vertically-spanned item begins at the top line of its range.

*Font changes*
— A key-letter 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, which is 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. The single letters `B`, `b`, `I`, and `i` are shorter synonyms for `fB` and `fI`. Font change commands

[2] More precisely, an *en* is a number of points (1 point = 1/72 inch) equal to half the current type size.

given with the table entries override these specifications.

*Point size changes*

— A key-letter 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 key-letter may be followed by the letter v or V and a number to indicate the vertical line spacing to be used within a multi-line 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 (see *Text Blocks* below).

*Column width indication*

— A key-letter 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 after the w, the largest element is considered 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; if none is used, the default is ens. If the width specification is a unitless integer, you may omit the parentheses. If the width value is changed in a column, the *last* one given controls.

*Equal width columns*

— A key-letter may be followed by the letter e or E to indicate equal width columns. All columns whose key-letters are followed by e or E are made the same width. In this way, you can format a group of regularly spaced columns.

Note:

The order of the above features is immaterial; they need not be separated by spaces, except as indicated above to avoid ambiguities involving point size and font changes. Thus a numerical column entry in italic font and 12-point type with a minimum width of 2.5 inches and separated by 6 ens from the next column could be specified as

```
np12w(2.5i)f I   6
```

*Alternative notation*

— Instead of listing the format of successive lines of a table on consecutive lines of the format section, separate successive line formats on the same line by commas. The format for the sample table above can be written:

```
   c s s, l n n .
```

*Default*
— Column descriptors missing from the end of a format line are assumed to be `L`. The longest line in the format section, however, defines the number of columns in the table; extra columns in the data are ignored silently.

## Data to be Formatted in the Table

Type the data for the table after the format line. Normally, each table line is typed as one line of data. Break very long input lines by typing a backslash ' \' as a continuation marker at the end of the run-on line. That line is combined with the following line upon formatting and the ' \' vanishes. The data for different columns, that is, the table entries, are separated by tabs, or by whatever character has been specified in the option *tabs* option. We recommend using a visible character such as the slash character '/'. There are a few special cases:

`troff` *commands within tables*
— An input line beginning with a ' . ' followed by anything except a digit is assumed to be a command to `troff` and is passed through unchanged, retaining its position in the table. So, for example, you can produce space within a table by `.sp` commands in the data.

*Full width horizontal lines*
— An input *line* containing only the character '_' (underscore) or '=' (equal sign) represents a single or double line, respectively, extending the full width of the *table*.

*Single column horizontal lines*
— An input table *entry* containing only the character '_' or '=' represents 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 '_' represents 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 a table format key-letter of '^'.

*Text blocks*
— To include a block of text as a table entry, precede it by `T{` and follow it by `T}`. To enter, as a single entry in the table, something that cannot conveniently be typed as a simple string between tabs, use:

```
. . . T{
block of text
T} . . .
```

Note that the `T}` end delimiter must begin a line; additional columns of data may follow after a tab on the same line. See the 'New York Area Rocks' example for an illustration of included text blocks in a table. If you use more than twenty or thirty text blocks in a table, various limits in the `troff` program are likely to be exceeded, producing diagnostics such as `too many text block diversions`.

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 $L \times C/(N+1)$ 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 (point size, font, etc.) used in setting the *block of text* are those in effect at the beginning of the table (including the effect of the `.TS` macro) and any table format specifications of size, spacing and font, using the `p`, `v` and `f` modifiers to the column key-letters. Commands within the text block itself are also recognized, of course. However, `troff` commands within the table data but not within the text block do not affect that block.

Note:

Although you can put any number of lines in a table, only the first 200 lines are used in calculating the widths of the various columns. Arrange a multi-page table 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, except for font and size changes indicated (a) in the table format section and (b) within the table data (as in the entry `\s+3\fIdata\fP\s0` ). Therefore, although arbitrary `troff` requests may be sprinkled in a table, use requests such as `.ps` (set the point size) with care to avoid confusing the width calculations.

### Changing the Format of a Table

If you must change the format of a table after many similar lines, as with sub-headings or summarizations, use the `.T&` (table continue) command to change column parameters. The outline of such a table input is:

```
.TS                            start of the table
options afecting the whole table    ;
format of the columns    .
data to be formatted in the table
.
.
.
data to be formatted in the table
.T&                            indicates a new format for the table
format of the columns    .
data to be formatted in the table
.
.
.
data to be formatted in the table
.T&                            indicates a new format for the table
format of the columns    .
data to be formatted in the table
.
.
.
data to be formatted in the table
.TE                            end of the table
```

as in the 'Composition of Foods' and 'Some London Transport Statistics' examples. Using this procedure, each table line can be close to its corresponding format line.

Note: 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.

## 6.3. Examples

Here are some examples illustrating features of `tbl`. Glance through them to find one that you can adapt to your needs.

Although you can use a tab to separate columns of data, a visible character is easier to read. The standard column separator here is the slash (`/`). If a slash is part of the data, we indicate a different separator, as in the first example.

**Input:**

```
.TS
tab (%) 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
tab (/) allbox;
c s s
c c c
n n n .
AT&T Common Stock
Year/Price/Dividend
1971/41-54/$2.60
2/41-54/2.70
3/46-55/2.87
4/40-53/3.24
5/45-52/3.40
6/51-59/.95*
.TE
* (first quarter only)
```

**Output:**

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

* (first quarter only)

**Input:**

```
.TS
tab (/) box;
c s s
c | c | c
l | l | n .
Major New York Bridges
=
Bridge/Designer/Length
_
Brooklyn/J. A. Roebling/1595
Manhattan/G. Lindenthal/1470
Williamsburg/L. L. Buck/1600
_
Queensborough/Palmer &/1182
/  Hornbostel
_
//1380
Triborough/O. H. Ammann/_
//383
_
Bronx Whitestone/O. H. Ammann/2300
Throgs Neck/O. H. Ammann/1800
_
George Washington/O. H. Ammann/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 &<br>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
tab (/) ;
c c
np-2 | n | .
/Stack
/_
1/46
/_
2/23
/_
3/15
/_
4/6.5
/_
5/2.1
/_
.TE
```

**Output:**

| | Stack |
|---|---|
| 1 | 46 |
| 2 | 23 |
| 3 | 15 |
| 4 | 6.5 |
| 5 | 2.1 |

**Input:**

```
.TS
tab (/) box;
L L L
L L _
L L | LB
L L _
L L L .
january/february/march
april/may
june/july/Months
august/september
october/november/december
.TE
```

**Output:**

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

**Input:**

```
.TS
 tab (/) box;
cfB s s s .
Composition of Foods
_
.T&
c  | c s s
c  | c s s
c   | c  | c  | c .
Food/Percent by Weight
\^/_
\^/Protein/Fat/Carbo-
\^/\^/\^/hydrate
_
.T&
l  | n  | n  | n .
Apples/.4/.5/13.0
Halibut/18.4/5.2/. . .
Lima beans/7.5/.8/22.0
Milk/3.3/4.0/5.0
Mushrooms/3.5/.4/6.0
Rye bread/9.0/.6/52.7
.TE
```

**Output:**

| **Composition of Foods** | | | |
|---|---|---|---|
| Food | Percent by Weight | | |
| | Protein | Fat | Carbohydrate |
| 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
tab (/) allbox;
cfI s s
c cw(1i) cw(1i)
lp9 lp9 lp9 .
New York Area Rocks
Era/Formation/Age (years)
Precambrian/Reading Prong/>1 billion
Paleozoic/Manhattan Prong/400 million
Mesozoic/T{
.na
Newark Basin, incl.
Stockton, Lockatong, and Brunswick
formations; also Watchungs
and Palisades.
T}/200 million
Cenozoic/Coastal Plain/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. |

**Input:**

```
.EQ
delim $$
.EN

. . .

.TS
tab (/) doublebox ;
c c
l l .
Name/Definition
.sp
.vs +2p
Gamma/$GAMMA (z) = int sub 0 sup inf  t sup {z-1} e sup -t dt$
Sine/$sin (x) = 1 over 2i ( e sup ix - e sup -ix )$
Error/$ roman erf (z) = 2 over sqrt pi int sub 0 sup z e sup {-t sup 2} dt$
Bessel/$ J sub 0 (z) = 1 over pi int sub 0 sup pi cos ( z sin theta ) d theta$
Zeta/$ zeta (s) = sum from k=1 to inf k sup -s ~~( Re~s > 1)$
.vs -2p
.TE
```

**Output:**

| Name | Definition |
|---|---|
| Gamma | $\Gamma(z)=\int_0^\infty t^{z-1}e^{-t}dt$ |
| Sine | $\sin(x)=\frac{1}{2i}(e^{ix}-e^{-ix})$ |
| Error | $\mathrm{erf}(z)=\frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$ |
| Bessel | $J_0(z)=\frac{1}{\pi}\int_0^\pi \cos(z\sin\theta)d\theta$ |
| Zeta | $\zeta(s)=\sum_{k=1}^\infty k^{-s}$ (Re $s$>1) |

**Input:**

```
.TS
box, tab (:) ;
cb s s s s
cp-2 s s s s
c | | c | c | c | c
c | | c | c | c | c
r2 | | n2 | n2 | n2 | n .
Readability of Text
Line Width & Leading for 10-Pt. Type
=
Line:Set:1-Point:2-Point:4-Point
Width:Solid:Leading:Leading:Leading
_
9 Pica:\-9.3:\-6.0:\-5.3:\-7.1
14 Pica:\-4.5:\-0.6:\-0.3:\-1.7
19 Pica:\-5.0:\-5.1: 0.0:\-2.0
31 Pica:\-3.7:\-3.8:\-2.4:\-3.6
43 Pica:\-9.1:\-9.0:\-5.9:\-8.8
.TE
```

**Output:**

| **Readability of Text** | | | | |
|---|---|---|---|---|
| Line Width & Leading for 10-Pt. Type | | | | |
| Line | Set | 1-Point | 2-Point | 4-Point |
| Width | Solid | Leading | Leading | Leading |
| 9 Pica | –9.3 | –6.0 | –5.3 | –7.1 |
| 14 Pica | –4.5 | –0.6 | –0.3 | –1.7 |
| 19 Pica | –5.0 | –5.1 | 0.0 | –2.0 |
| 31 Pica | –3.7 | –3.8 | –2.4 | –3.6 |
| 43 Pica | –9.1 | –9.0 | –5.9 | –8.8 |

**Input:**

```
.TS
tab (/) ;
c s
cip-2 s
l n
a n .
Some London Transport Statistics
(Year 1964)
Railway route miles/244
Tube/66
Sub-surface/22
Surface/156
.sp .5
.T&
l r
a r .
Passenger traffic \- railway
Journeys/674 million
Average length/4.55 miles
Passenger miles/3,066 million
.T&
l r
a r .
Passenger traffic \- road
Journeys/2,252 million
Average length/2.26 miles
Passenger miles/5,094 million
.T&
l n
a n .
.sp .5
Vehicles/12,521
Railway motor cars/2,905
Railway trailer cars/1,269
Total railway/4,174
Omnibuses/8,347
.T&
l n
a n .
.sp .5
Staff/73,739
Administrative, etc./8,553
Civil engineering/5,134
Electrical eng./1,714
Mech. eng. \- railway/4,310
Mech. eng. \- road/9,152
Railway operations/8,930
Road operations/35,946
.TE
```

## Output:

Some London Transport Statistics

*(Year 1964)*

| | |
|---|---|
| Railway route miles | 244 |
| Tube | 66 |
| Sub-surface | 22 |
| Surface | 156 |
| Passenger traffic – railway | |
| Journeys | 674 million |
| Average length | 4.55 miles |
| Passenger miles | 3,066 million |
| Passenger traffic – road | |
| Journeys | 2,252 million |
| Average length | 2.26 miles |
| Passenger miles | 5,094 million |
| Vehicles | 12,521 |
| Railway motor cars | 2,905 |
| Railway trailer cars | 1,269 |
| Total railway | 4,174 |
| Omnibuses | 8,347 |
| Staff | 73,739 |
| Administrative, etc. | 5,553 |
| Civil engineering | 5,134 |
| Electrical eng. | 1,714 |
| Mech. eng. – railway | 4,310 |
| Mech. eng. – road | 9,152 |
| Railway operations | 8,930 |
| Road operations | 35,946 |

**Input:**

```
.ps 8
.vs 10p
.TS
tab (/) center box;
c s s
ci s s
c c c
lB l n .
New Jersey Representatives
(Democrats)
.sp .5
Name/Office address/Phone
.sp .5
James J. Florio/23 S. White Horse Pike, Somerdale 08083/609-627-8222
William J. Hughes/2920 Atlantic Ave., Atlantic City 08401/609-345-4844
James J. Howard/801 Bangs Ave., Asbury Park 07712/201-774-1600
Frank Thompson, Jr./10 Rutgers Pl., Trenton 08618/609-599-1619
Andrew Maguire/115 W. Passaic St., Rochelle Park 07662/201-843-0240
Robert A. Roe/U.S.P.O., 194 Ward St., Paterson 07510/201-523-5152
Henry Helstoski/666 Paterson Ave., East Rutherford 07073/201-939-9090
Peter W. Rodino, Jr./Suite 1435A, 970 Broad St., Newark 07102/201-645-3213
Joseph G. Minish/308 Main St., Orange 07050/201-645-6363
Helen S. Meyner/32 Bridge St., Lambertville 08530/609-397-1830
Dominick V. Daniels/895 Bergen Ave., Jersey City 07306/201-659-7700
Edward J. Patten/Natl. Bank Bldg., Perth Amboy 08861/201-826-4610
.sp .5
.T&
ci s s
lB l n .
(Republicans)
.sp .5v
Millicent Fenwick/41 N. Bridge St., Somerville 08876/201-722-8200
Edwin B. Forsythe/301 Mill St., Moorestown 08057/609-235-6622
Matthew J. Rinaldo/1961 Morris Ave., Union 07083/201-687-4235
.TE
.ps 10
.vs 12p
```

**Output:**

| New Jersey Representatives | | |
|---|---|---|
| *(Democrats)* | | |
| Name | Office address | Phone |
| **James J. Florio** | 23 S. White Horse Pike, Somerdale 08083 | 609-627-8222 |
| **William J. Hughes** | 2920 Atlantic Ave., Atlantic City 08401 | 609-345-4844 |
| **James J. Howard** | 801 Bangs Ave., Asbury Park 07712 | 201-774-1600 |
| **Frank Thompson, Jr.** | 10 Rutgers Pl., Trenton 08618 | 609-599-1619 |
| **Andrew Maguire** | 115 W. Passaic St., Rochelle Park 07662 | 201-843-0240 |
| **Robert A. Roe** | U.S.P.O., 194 Ward St., Paterson 07510 | 201-523-5152 |
| **Henry Helstoski** | 666 Paterson Ave., East Rutherford 07073 | 201-939-9090 |
| **Peter W. Rodino, Jr.** | Suite 1435A, 970 Broad St., Newark 07102 | 201-645-3213 |
| **Joseph G. Minish** | 308 Main St., Orange 07050 | 201-645-6363 |
| **Helen S. Meyner** | 32 Bridge St., Lambertville 08530 | 609-397-1830 |
| **Dominick V. Daniels** | 895 Bergen Ave., Jersey City 07306 | 201-659-7700 |
| **Edward J. Patten** | Natl. Bank Bldg., Perth Amboy 08861 | 201-826-4610 |
| *(Republicans)* | | |
| **Millicent Fenwick** | 41 N. Bridge St., Somerville 08876 | 201-722-8200 |
| **Edwin B. Forsythe** | 301 Mill St., Moorestown 08057 | 609-235-6622 |
| **Matthew J. Rinaldo** | 1961 Morris Ave., Union 07083 | 201-687-4235 |

This is a paragraph of normal text placed here only to indicate where the left and right margins are. Examine the appearance of centered tables or expanded tables, and observe how such tables are formatted.

**Input:**

```
.TS
center tab (/) ;
c s s s
c s s s
c c c c
n n n n .
LYKE WAKE WALK
Successful Crossings 1959-1966
Year/First Crossings/Repeats/Total
1959/89/23/112
1960/222/33/255
1961/650/150/800
1962/1100/267/1367
1963/1054/409/1463
1964/1413/592/2005
1965/2042/771/2813
1966/2537/723/3260
 .TE
```

**Output:**

LYKE WAKE WALK

Successful Crossings 1959–1966

| Year | First Crossings | Repeats | Total |
|---|---|---|---|
| 1959 | 89 | 23 | 112 |
| 1960 | 222 | 33 | 255 |
| 1961 | 650 | 150 | 800 |
| 1962 | 1100 | 267 | 1367 |
| 1963 | 1054 | 409 | 1463 |
| 1964 | 1413 | 592 | 2005 |
| 1965 | 2042 | 771 | 2813 |
| 1966 | 2537 | 723 | 3260 |

**Input:**

```
.TS
tab (/) box;
cb   s   s   s
c | c | c   s
ltiw(1i) | ltw(2i) | lp8 | lw(1.6i)p8 .
Some Interesting Places
_
Name/Description/Practical Information
_
T{
American Museum of Natural History
T}/T{
The collections fill 11.5 acres (Michelin) or 25 acres (MTA)
of exhibition halls on four floors.  There is a full-sized replica
of a blue whale and the world's largest star sapphire (stolen in 1964).
T}/Hours/10-5, ex. Sun 11-5, Wed. to 9
\^/\^/Location/T{
Central Park West & 79th St.
T}
\^/\^/Admission/Donation: $1.00 asked
\^/\^/Subway/AA to 81st St.
\^/\^/Telephone/212-873-4225
_
Bronx Zoo/T{
About a mile long and .6 mile wide, this is the largest zoo in America.
A lion eats 18 pounds
of meat a day while a sea lion eats 15 pounds of fish.
T}/Hours/T{
10-4:30 winter, to 5:00 summer
T}
\^/\^/Location/T{
185th St. & Southern Blvd, the Bronx.
T}
\^/\^/Admission/$1.00, but Tu,We,Th free
\^/\^/Subway/2, 5 to East Tremont Ave.
\^/\^/Telephone/212-933-1759
_
Brooklyn Museum/T{
Five floors of galleries contain American and ancient art.
There are American period rooms and architectural ornaments saved
from wreckers, such as a classical figure from Pennsylvania Station.
T}/Hours/Wed-Sat, 10-5, Sun 12-5
\^/\^/Location/T{
Eastern Parkway & Washington Ave., Brooklyn.
T}
\^/\^/Admission/Free
\^/\^/Subway/2,3 to Eastern Parkway.
\^/\^/Telephone/212-638-5000
_
T{
New-York Historical Society
T}/T{
All the original paintings for Audubon's
.I
Birds of America
.R
are here, as are exhibits of American decorative arts, New York history,
Hudson River school paintings, carriages, and glass paperweights.
T}/Hours/T{
Tues-Fri & Sun, 1-5; Sat 10-5
T}
\^/\^/Location/T{
Central Park West & 77th St.
T}
\^/\^/Admission/Free
\^/\^/Subway/AA to 81st St.
\^/\^/Telephone/212-873-3400
.TE
```

**Output:**

| Some Interesting Places | | | |
|---|---|---|---|
| Name | Description | Practical Information | |
| *American Museum of Natural History* | The collections fill 11.5 acres (Michelin) or 25 acres (MTA) of exhibition halls on four floors. There is a full-sized replica of a blue whale and the world's largest star sapphire (stolen in 1964). | Hours<br>Location<br>Admission<br>Subway<br>Telephone | 10-5, ex. Sun 11-5, Wed. to 9<br>Central Park West & 79th St.<br>Donation: $1.00 asked<br>AA to 81st St.<br>212-873-4225 |
| *Bronx Zoo* | About a mile long and .6 mile wide, this is the largest zoo in America. A lion eats 18 pounds of meat a day while a sea lion eats 15 pounds of fish. | Hours<br>Location<br>Admission<br>Subway<br>Telephone | 10-4:30 winter, to 5:00 summer<br>185th St. & Southern Blvd, the Bronx.<br>$1.00, but Tu,We,Th free<br>2, 5 to East Tremont Ave.<br>212-933-1759 |
| *Brooklyn Museum* | Five floors of galleries contain American and ancient art. There are American period rooms and architectural ornaments saved from wreckers, such as a classical figure from Pennsylvania Station. | Hours<br>Location<br>Admission<br>Subway<br>Telephone | Wed-Sat, 10-5, Sun 12-5<br>Eastern Parkway & Washington Ave., Brooklyn.<br>Free<br>2,3 to Eastern Parkway.<br>212-638-5000 |
| *New-York Historical Society* | All the original paintings for Audubon's *Birds of America* are here, as are exhibits of American decorative arts, New York history, Hudson River school paintings, carriages, and glass paperweights. | Hours<br>Location<br>Admission<br>Subway<br>Telephone | Tues-Fri & Sun, 1-5; Sat 10-5<br>Central Park West & 77th St.<br>Free<br>AA to 81st St.<br>212-873-3400 |

## 6.4. `tbl` Commands

Table 6-1 `tbl` *Command Characters and Words*

| *Command* | *Meaning* |
|---|---|
| `a A` | Alphabetic subcolumn |
| `allbox` | Draw box around all items |
| `b B` | Boldface item |
| `box` | Draw box around table |
| `c C` | Centered column |
| `center` | Center table in page |
| `doublebox` | Doubled box around table |
| `e E` | Equal width columns |
| `expand` | Make 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 |
| `T{    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 |
| _ | Horizontal line |
| _ | Short horizontal line |

# 7

# Typesetting Mathematics with eqn

7

# Typesetting Mathematics with eqn

This chapter explains how to use the eqn preprocessor for printing mathematics on a phototypesetter, and provides numerous examples after which to model equations in your documents.[1]

You describe mathematical expressions in an English-like language that the eqn program translates into troff commands for final troff formatting. In other words, eqn sets the mathematics while troff does the body of the text. eqn provides accurate and relatively easy mathematical phototypesetting, which is not easy to accomplish with normal typesetting machines. Because the mathematical expressions are embedded in the running text of a manuscript, the entire document is produced in one process. For example, you can set in-line expressions like $\lim_{x \to \pi/2} (\tan x)^{\sin 2x} = 1$ or display equations like

$$
\begin{aligned}
G(z) = e^{\ln G(z)} = \exp\left( \sum_{k \ge 1} \frac{S_k z^k}{k} \right) &= \prod_{k \ge 1} e^{S_k z^k / k} \\
&= \left( 1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots \right) \left( 1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \cdots \right) \cdots \\
&= \sum_{m \ge 0} \left( \sum_{\substack{k_1, k_2, \ldots, k_m \ge 0 \\ k_1 + 2k_2 + \cdots + mk_m = m}} \frac{S_1^{k_1}}{1^{k_1} k_1!} \, \frac{S_2^{k_2}}{2^{k_2} k_2!} \cdots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right) z^m
\end{aligned}
$$

eqn knows relatively little about mathematics. In particular, mathematical symbols like +, –, ×, parentheses, and so on have no special meanings. eqn is quite happy to set these symbols, and they will look good.

eqn also produces mathematics with nroff. The input is identical, but you have to use the programs neqn and nroff instead of eqn and troff. Of course, some things won't look as good because your workstation or terminal does not provide the variety of characters, sizes and fonts that a phototypesetter does, but the output is usually adequate for proofreading.

[1] The material in this chapter is derived from *A System for Typesetting Mathematics*, B.W. Kernighan, L. L. Cherry and *Typesetting Mathematics — User's Guide*, B.W. Kernighan, L.L. Cherry, Bell Laboratories, Murray Hill, New Jersey.

## 7.1. Displaying Equations — .EQ and .EN

To tell eqn where a mathematical expression begins and ends, mark it with lines beginning .EQ and .EN. Thus if you type the lines:

```
.EQ
x=y+z
.EN
```

your output will look like:

$$x=y+z$$

eqn copies '.EQ' and '.EN' through untouched. This means that you have to take care of things like centering, numbering, and so on yourself. The common way is to use the troff and nroff macro package package '**-ms**', which provides macros for centering, indenting, left-justifying and making numbered equations.

With the **-ms** package, equations are centered by default. To left-justify an equation, use .EQ L instead of .EQ. To indent it, use .EQ I.

You can also supplement eqn with troff commands as desired; for example, you can produce a centered display with the input:

```
.ce
.EQ
x sub i = y sub i ...
.EN
```

which produces

$$x_i=y_i \cdots$$

You can call out any of these by an arbitrary 'equation number,' which will be placed at the right margin. For example, the input

```
.EQ I (3.1a)
x = f(y/2) + y/2
.EN
```

produces the output

$$x=f\,(y/2)+y/2 \qquad (3.1a)$$

There is also a shorthand notation so you can enter in-line expressions like $\pi_i^2$ without .EQ and .EN. This is described in the section "Shorthand for In-line Equations."

## 7.2. Running eqn and neqn

To print a document that contains mathematics on the phototypesetter, use:

```
hostname% eqn files | troff -options | lpr -t -Printer
```

troff or your installation's equivalent does the formatting, which is sent to your phototypesetter as indicated by -P*printer*. If you use the **-ms** macro package for example, type:

```
hostname% eqn files | troff -ms -t | lpr -t -Printer
```

To display equations on the standard output, your workstation screen, use nroff as follows:

```
hostname% neqn files | nroff -options
```

The language for equations recognized by neqn is identical to that of eqn, although of course the output is more restricted. You can use the online rendition of the mathematical formulae for proofing, but the output does not accurately represent the symbols and fonts. You can of course pipe the output through more for easier viewing:

```
hostname% neqn files | nroff -options | more
```

or redirect it to a file:

```
hostname% neqn files | nroff -options > newfile
```

To use a GSI or DASI terminal as the output device, type:

```
hostname% neqn files | nroff -Tx
```

where *x* is the terminal type you are using, such as *300* or *300S*. To send neqn output to the printer, type:

```
hostname% neqn file | nroff -options | lpr -Pprinter
```

You can use eqn and neqn with the tbl program for setting tables that contain mathematics. Use tbl before eqn or neqn, like this:

```
hostname% tbl  files | eqn  | troff -options
```

or

```
hostname% tbl  files | neqn  | nroff -options
```

## 7.3. Putting Spaces in the Input Text

eqn removes spaces and newlines within an expression and leaves normal text alone. Thus, between .EQ and .EN,

```
.EQ
x=y+z
.EN
```

and

```
.EQ
x = y + z
.EN
```

and

```
.EQ
x    =    y
    + z
.EN
```

all produce the same output, namely:

$$x=y+z$$

You should use spaces and newlines freely to make your input equations readable and easy to edit. In particular, very long lines are a bad idea, since they are often hard to fix if you make a mistake.

The only way eqn can deduce that some sequence of letters might be special is if that sequence is separated from the letters on either side of it. To do this, surround a special word by ordinary spaces (or tabs or newlines), as shown in the previous section.

You can also make special words stand out by surrounding them with tildes or circumflexes:

```
.EQ
x~=~2~pi~int~sin~(~omega~t~)~dt
.EN
```

is much the same as the last example, except that the tildes not only separate the magic words like sin, omega, and so on, but also add extra spaces, one space per tilde:

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

You can also use braces { } and double quotes " . . . " to separate special words; these characters that have special meanings are described later.

Remembering that a blank is a delimiter can be a problem. For instance, a common mistake is typing:

```
.EQ
f(x sub i)
.EN
```

which produces

$$f(x_{i)}$$

instead of

$$f(x_i)$$

eqn cannot tell that the right parenthesis is not part of the subscript. Type instead:

```
.EQ
f(x sub i )
.EN
```

## 7.4. Producing Spaces in the Output Text

To force extra spaces into the output, use a tilde ~ for each space you want:

```
.EQ
x~=~y~+~z
.EN
```

gives

$$x ~=~ y ~+~ z$$

You can also use a circumflex ^, which gives a space half the width of a tilde. It is mainly useful for fine-tuning. Use tabs to position pieces of an expression, but you must use troff commands to set the tab stops.

## 7.5. Symbols, Special Names, and Greek Letters

eqn knows some mathematical symbols, some mathematical names, and the Greek alphabet. For example,

```
.EQ
x=2 pi int sin ( omega t)dt
.EN
```

produces

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

Here the spaces in the input are **necessary** 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 made Greek; and int becomes the integral sign.

When in doubt, leave spaces around separate parts of the input. A *very* common error is to type

```
f(pi)
```

without leaving spaces on both sides of the `pi`. As a result, `eqn` does not recognize `pi` as a special word, and it appears as $f(pi)$ instead of $f(\pi)$.

A complete list of `eqn` names appears in the section "Precedences and Keywords." You can also use special characters available in `troff` for anything `eqn` doesn't know about.

### 7.6. Subscripts and Superscripts — `sub` and `sup`

To obtain subscripts and superscripts, use the words `sub` and `sup`.

```
.EQ
x sup 2 + y sub k
.EN
```

gives

$$x^2+y_k$$

`eqn` takes care of all the size changes and vertical motions needed to make the output look right. You must surround the words `sub` and `sup` by spaces; *x sub2* gives you $xsub2$ instead of $x_2$. As another example, consider:

```
.EQ
x sup 2 + y sup 2 = z sup 2
.EN
```

which produces:

$$x^2+y^2=z^2$$

Furthermore, don't forget to leave a space (or a tilde, etc.) to mark the end of a subscript or superscript. A common error is to say something like

```
.EQ
y = (x sup 2)+1
.EN
```

which causes

$$y=(x^{2)+1}$$

instead of the intended

$$y=(x^2)+1$$

which is produced by:

```
.EQ
y = (x sup 2 )+1
.EN
```

Subscripted subscripts and superscripted superscripts also work:

```
.EQ
x sub i sub 1
.EN
```

is

$$x_{i_1}$$

A subscript and superscript on the same thing are printed one above the other if the subscript comes first:

```
.EQ
x sub i sup 2
.EN
```

is

$$x_i^2$$

Other than this special case, `sub` and `sup` group to the right, so

```
x sup y sub z
```

means $x^{y_z}$, not ${x^y}_z$.

## 7.7. Grouping Equation Parts — { and }

Normally, the end of a subscript or superscript is marked simply by a blank, tab, tilde, and so on. If the subscript or superscript is something that has to be typed with blanks in it, use the braces { and } to mark the beginning and end of the subscript or superscript:

```
.EQ
e sup {i omega t}
.EN
```

is

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

You can *always* use braces to force `eqn` to treat something as a unit, or just to make your intent perfectly clear. Thus:

```
.EQ
x sub {i sub 1} sup 2
.EN
```

is

$$x_{i_1}^2$$

with braces, but

```
.EQ
x sub i sub 1 sup 2
.EN
```

is

$$x_{i_1^2}$$

which is rather different.

Braces can occur within braces if necessary:

```
.EQ
e sup {i pi sup {rho +1}}
.EN
```

is

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

The general rule is that anywhere you could use some single entry like $x$, you can use an arbitrarily complicated entry if you enclose it in braces. `eqn` looks after all the details of positioning it and making it the right size.

In all cases, make sure you have the right number of braces. Leaving one out or adding an extra causes `eqn` to complain bitterly.

Occasionally you have to print braces. To do this, enclose them in double quotes, like `"{"`. Quoting is discussed in more detail in *Quoted Text.*

## 7.8. Fractions — `over`

To make a fraction, use the word `over`:

```
.EQ
a+b over 2c =1
.EN
```

gives

$$\frac{a+b}{2c}=1$$

The line is made the right length and positioned automatically.

```
.EQ
a+b over c+d+e = 1
.EN
```

produces

$$\frac{a+b}{c+d+e}=1$$

Use braces to clarify what goes over what:

```
.EQ
{alpha + beta} over {sin (x)}
.EN
```

is

$$\frac{\alpha+\beta}{\sin(x)}$$

When there is both an over and a sup in the same expression, eqn does the sup before the over, so

```
.EQ
-b sup 2 over pi
.EN
```

is $\frac{-b^2}{\pi}$ instead of $-b^{\frac{2}{\pi}}$ The rules that determine which operation is done first in cases like this are summarized in the section "Precedences and Keywords." When in doubt, however, use braces to make clear what goes with what.

## 7.9. Square Roots — sqrt

To draw a square root, use sqrt:

```
.EQ
sqrt a+b
.EN
```

produces

$$\sqrt{a+b}$$

and

```
.EQ
sqrt a+b + 1 over sqrt {ax sup 2 +bx+c}
.EN
```

is

$$\sqrt{a+b}+\frac{1}{\sqrt{ax^2+bx+c}}$$

Note: Square roots of tall quantities look sloppy because a root-sign big enough to cover the quantity is too dark and heavy:

```
.EQ
sqrt {a sup 2 over b sub 2}
.EN
```

is

$$\sqrt{\frac{a^2}{b_2}}$$

Big square roots are generally better written as something to a power:

$$(a^2/b_2)^{\frac{1}{2}}$$

which is

```
.EQ
(a sup 2 /b sub 2 ) sup {1 over 2}
.EN
```

## 7.10. Summation, Integral, and Other Large Operators

To produce summations, integrals, and similar constructions, use:

```
.EQ
sum from i=0 to {i= inf} x sub i
.EN
```

which produces

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

Notice that you use braces to indicate where the upper part $i=\infty$ begins and ends. No braces are necessary for the lower part $i=0$, because it does not contain any blanks. The braces will never hurt, and if the `from` and `to` parts contain any blanks, you must use braces around them.

The `from` and `to` parts are both optional, but if both are used, they have to occur in that order.

Other useful characters can replace the `sum` in our example:

```
.EQ
int   prod   union   inter
.EN
```

become, respectively,

$$\int \quad \prod \quad \cup \quad \cap$$

Since the thing before the `from` can be anything, even something in braces, `from-to` can often be used in unexpected ways:

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

is

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

## 7.11. Size and Font Changes

By default, equations are set in 10-point type with standard mathematical conventions to determine what characters are in roman and what in italic. Although eqn makes a valiant attempt to use aesthetically pleasing sizes and fonts, it is not perfect. To change sizes and fonts, use `size` *n* and `roman`, `italic`, `bold` and `fat`. Like `sub` and `sup`, size and font changes affect only the thing that follows them; they revert to the normal situation at the end of it. Thus

```
.EQ
bold x y
.EN
```

is

$$\mathbf{xy}$$

and

```
.EQ
size 14 bold x = y +
   size 14 {alpha + beta}
.EN
```

gives

$$\mathbf{x} = y + \alpha + \beta$$

As always, you can use braces if you want to affect something more complicated than a single letter. For example, you can change the size of an entire equation by

```
.EQ
size 12 { ... }
.EN
```

Legal sizes that may follow `size` are the same as those allowed in `troff`: 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, 36. You can also change the size by a given increment or decrement. For example, you can say `size +2` to make the size two points bigger, or `size -3` to make it three points smaller. This is easier because you don't have to know what the current size is.

The `size` variable in eqn translates into a `troff` `\s` construct. However, `troff` only recognizes one digit after the + or – sign. Therefore, `\s+9` or `\s-9` are respectively the largest incremental and decremental point size changes.

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. Since eqn is tuned for roman, italic and bold, other fonts may not appear quite as good.

The `fat` operation takes the current font and widens it by overstriking: `fat grad` is $\boldsymbol{\nabla}$ and `fat {x sub i}` is $\boldsymbol{x_i}$.

If an entire document is to be in a non-standard size or font, it is a severe nuisance to have to write out a size and font change for each equation. Accordingly, 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 and the font to roman thereafter. In place of `R`, you can use any of the `troff` font names. 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: you can change the global font and size as often as needed. For example, in a footnote[2] 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.

## 7.12. Diacritical Marks

To get accent marks on top of letters, there are several words:

| | |
|---|---|
| `x dot` | $\dot{x}$ |
| `x dotdot` | $\ddot{x}$ |
| `x hat` | $\hat{x}$ |
| `x tilde` | $\tilde{x}$ |
| `x vec` | $\vec{x}$ |
| `x dyad` | $\overleftrightarrow{x}$ |
| `x bar` | $\bar{x}$ |
| `x under` | $\underline{x}$ |

The diacritical mark is placed at the right height. The `bar` and `under` are made the right length for the entire construct, as in $\overline{x+y+z}$; other marks are centered. For example

[2] Like this one, in which we have a few random expressions like $x_i$ and $\pi^2$. The sizes for these were set by the command `gsize~-2`.

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

produces

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

## 7.13. Quoted Text

Any input entirely within quotes ( "..." ) is not subject to any of the font changes and spacing adjustments that you normally set. This provides a way to do your own spacing and adjusting if needed:

```
.EQ
italic "sin(x)" + sin (x)
.EN
```

is

$$\mathit{sin(x)}+\sin(x)$$

You also use quotes to get braces and other eqn keywords printed:

```
.EQ
"{ size alpha }"
.EN
```

is

*{ size alpha }*

and

```
.EQ
roman "{ size alpha }"
.EN
```

is

{ size alpha }

The construction " " is often used as a place-holder when grammatically eqn needs something, but you don't actually want anything in your output. For example, to make $^2$He, you can't just type `sup 2 roman He` because a `sup` has to be a superscript *on* something. Thus you must say

```
.EQ
"" sup 2 roman He
.EN
```

To get a literal quote use `\"`. `troff` characters like `\(bs` can appear unquoted, but more complicated things like horizontal and vertical motions with `\h` and `\v` should always be quoted.

## 7.14. Lining Up Equations — `mark` and `lineup`

Sometimes it's necessary to line up a series of equations at some horizontal position, often at an equals sign. To do this, use the 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

$$x+y=z$$

$$x=1$$

For reasons beyond the scope of this chapter, when you use `eqn` and **-ms**, use either `.EQ I` or `.EQ L`, as `mark` and `lineup` don't work with centered equations. Also bear in mind that `mark` doesn't look ahead;

```
.EQ
x mark =1
 ...
x+y lineup =z
.EN
```

isn't going to work, because there isn't room for the `x+y` part after the `mark` has processed the `x`.

## 7.15. Big Brackets

To get big brackets `[ ]`, braces `{ }`, parentheses `( )`, and bars `| |` around things, use the `left` and `right` commands:

```
.EQ
left { a over b + 1 right }
 ~=~ left ( c over d right )
 + left [ e right ]
.EN
```

is

$$\left\{ \frac{a}{b}+1 \right\} = \left[ \frac{c}{d} \right] + \left[ e \right]$$

The resulting brackets are made big enough to cover whatever they enclose. Other characters can be used besides these, but they are not likely to look very good. Two exceptions are the `floor` and `ceiling` characters:

```
.EQ
left floor x over y right floor
<= left ceiling a over b right ceiling
.EN
```

produces

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

Several warnings about brackets are in order. First, braces are typically bigger than brackets and parentheses, because they are made up of three, five, seven, etc., pieces, while brackets can be made up of two, three, etc. Second, big left and right parentheses often look poor, because the character set is poorly designed.

The `right` part may be omitted: a 'left something' need not have a corresponding 'right something'. 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 )
```

for example. The `left ""` means a 'left nothing'. This satisfies the rules without hurting your output.

### 7.16. Piles — `pile`

There is a general facility for making vertical piles of things; it comes in several flavors. For example:

```
.EQ
A ~=~ left [
  pile { a above b above c }
  ~~ pile { x above y above z }
right ]
.EN
```

will make

$$A = \left[ \begin{matrix} a & x \\ b & y \\ c & z \end{matrix} \right]$$

The elements of the pile are centered one above another at the right height for most purposes. There can be as many elements as you want. The keyword `above` is used to separate the pieces; put braces around the entire list. The elements of a pile can be as complicated as needed, even containing more piles.

Three other forms of pile exist: `lpile` makes a pile with the elements left-justified; `rpile` makes a right-justified pile; and `cpile` makes a centered pile, just like `pile`. The vertical spacing between the pieces is somewhat larger for `lpiles`, `rpiles`, and `cpiles` than it is for ordinary piles. For example:

```
.EQ
roman sign (x)~=~
left {
   lpile {1 above 0 above -1}
   ~~ lpile
    {if~x>0 above if~x=0 above if~x<0}
.EN
```

makes

$$\mathrm{sign}(x) = \left\{ \begin{array}{ll} 1 & \mathrm{if}\, x>0 \\ 0 & \mathrm{if}\, x=0 \\ -1 & \mathrm{if}\, x<0 \end{array} \right.$$

Notice the left brace without a matching right one.

### 7.17. Matrices — `matrix`

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

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

you have to type

```
.EQ
matrix {
  ccol { x sub i above y sub i }
  ccol { x sup 2 above y sup 2 }
}
.EN
```

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, by the way, is that if the elements of the piles don't all have the same height, they won't 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.** Otherwise, results are unpredictable.

## 7.18. Shorthand for In-line Equations — `delim`

In a mathematical document, it is necessary to follow mathematical conventions not just in display equations, but also in the body of the text. For example you need variable names like *x* to be in italics. Although you can do this by surrounding the appropriate parts with the macro requests `.EQ` and `.EN`, the continual repetition of `.EQ` and `.EN` is a nuisance. Furthermore, with **-ms**, `.EQ` and `.EN` imply 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 in the middle of text lines. To set both the left and right characters to dollar signs, for example, add to the beginning of your document the three lines

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

Having done this, you can then say things like

```
Let $alpha sub i$ be the primary variable, and let $beta$
be zero.  Then we can show that $x sub 1$ is $>=0$.
```

This works as you might expect; spaces, 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 $sum from i=1 to n x sub i$ does not interfere with the lines surrounding it.

The printed result looks like: Let $\alpha_i$ be the primary variable, and let $\beta$ be zero. Then we can show that $x_1$ is $\geq 0$.

To turn off the delimiters, use:

```
.EQ
delim off
.EN
```

Notes: Don't use braces, tildes, circumflexes, or double quotes as delimiters; chaos will result. Also, if you're using `tbl`, don't use sharps (pound signs) either.

## 7.19. Definitions — `define`

`eqn` provides a string-naming facility so you can 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

```
.EQ
x sub i sub 1 + y sub i sub 1
.EN
```

appears repeatedly throughout a paper, you can save re-typing it each time by defining it like this:

```
.EQ
define  xy  'x sub i sub 1 + y sub i sub 1'
.EN
```

This makes xy a shorthand for whatever characters occur between the single quotes in the definition. You can use any character instead of quote to mark the ends of the definition, as long as it doesn't appear inside the definition.

Now you can use xy like this:

```
.EQ
f(x) = xy ...
.EN
```

and so on. Each occurrence of xy will expand into what it was defined as. Be sure 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.** A common error is to say

```
.EQ
define  X  ' roman X '
.EN
```

This is a guaranteed disaster, since X is now defined in terms of itself. If you say

```
.EQ
define  X  ' roman "X" '
.EN
```

however, the quotes protect the second X, and everything works fine.

You can redefine eqn keywords. You can make slash (/) mean over by saying

```
.EQ
define  /  ' over '
.EN
```

or redefine `over` as / with

```
.EQ
define  over  ' / '
.EN
```

If you need things to print on a workstation or terminal as well as on the phototypesetter, it is sometimes worth defining a symbol differently in `neqn` and `eqn`. To do this, use `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 plain `define` apply to both `eqn` and `neqn`.

## 7.20. Tuning the Spacing

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. You can get small extra horizontal spaces with tilde and circumflex. You can also say `back` *n* and `fwd` *n* to move small amounts horizontally. The *n* is how far to move in 1/100s of an em (an em is about the width of the letter 'm'.) Thus `back 50` moves back about half the width of an m. Similarly you can move things up or down with `up` *n* and `down` *n*. As with `sub` or `sup`, the local motions affect the next thing in the input, and this can be anything if it is enclosed in braces.

## 7.21. Troubleshooting

If you make a mistake in an equation, like leaving out a brace, having one too many, or having a `sup` with nothing before it, `eqn` tells you with the message:

```
syntax error between lines x and y, file z
```

where *x* and *y* are approximately the lines between which the trouble occurred, and *z* is the name of the file in question. The line numbers are approximate, so look nearby as well. There are also self-explanatory messages that arise if you leave out a quote or try to run `eqn` on a non-existent file.

If you want to check a document before actually printing it, run:

```
hostname% eqn files >/dev/null
```

to throw away the output but display the messages.

If you use something like dollar signs as delimiters, it is easy to leave one out. You may also occasionally forget one half of a pair of macros or have an unbalanced font change. These can cause problems, but you can check for balanced pairs of delimiters and macros with `checkeq` and `checknr`. For instance, to run `checkeq` on this chapter called `eqn.ug` to check for unbalanced pairs of `.EQ`s and `.EN`s, type:

```
hostname% checkeq eqn.ug
eqn.ug:
   New delims , line 2
    in EQ, line 2
   Spurious EN, line 46
   Delim off, line 1254
   New delims , line 1278
   New delims , line 1635
    in EQ, line 1635
   New delims ##, line 1991
   Delim off, line 1999
hostname%
```

We left out the `.EQ` before the `.EN` on line 46 to show you some sample output. This also reports on the delimiters. You can also use `checknr` with specific options to check specifically for a particular macro pair. For example, to run `checknr` to check that there is an `.EQ` for every `.EN`, type:

```
hostname% checknr -s -f -a.EQ.EN eqn.ug
46: Unmatched .EN
hostname%
```

Specify the macro pair you want to check for with the **–a** option and the six characters in the pair. The `-s` option ignores size changes and the `-f` option ignores font changes. See `checknr(1)` in the *SunOS Reference Manual* for more details.

Inline equations can only be so big because of an internal buffer in `troff`. If you get a message `word overflow`, you have exceeded this limit. If you print the equation as a displayed equation, that is, offset from the body of the text with `.EQ` and `.EN`, 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.

On a related topic, `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` does warn about equations that are too long to fit on one line.

## 7.22. Precedences and Keywords

If you don't use braces, `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

The operations that group to the left are:

> over sqrt left right

All others group to the right. For example, in the expression

```
.EQ
a sup 2 over b
.EN
```

`sup` is defined to have a higher precedence than `over`, so this construction is parsed as $\frac{a^2}{b}$ instead of $a^{\frac{2}{b}}$. Naturally, you can always force a particular parsing by placing braces around expressions.

Digits, parentheses, brackets, punctuation marks, and the following 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
```

The following character sequences are recognized and translated as shown.

Table 7-1 *Character Sequence Translation*

| You Type | Translation |
|---|---|
| `>=` | ≥ |
| `<=` | ≤ |
| `==` | ≡ |
| `!=` | ≠ |
| `+-` | ± |
| `->` | → |
| `<-` | ← |
| `<<` | ≪ |
| `>>` | ≫ |
| `inf` | ∞ |
| `partial` | ∂ |
| `prime` | ′ |
| `approx` | ≈ |
| `nothing` | |
| `cdot` | · |
| `times` | × |
| `del` | ∇ |
| `grad` | ∇ |
| `...` | ⋯ |
| `,...,` | ,⋯, |
| `sum` | Σ |
| `int` | ∫ |
| `prod` | Π |
| `union` | ∪ |
| `inter` | ∩ |

To obtain Greek letters, simply spell them out in whatever case you want:

Table 7-2 *Greek Letters*

| You Type | Translation | You Type | Translation |
|---|---|---|---|
| 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 | ο |
| 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 | | |

The eqn keywords, except for characters with names, follow.

Table 7-3 eqn *Keywords*

above
back
bar
bold
ccol
col
cpile
define
delim
dot
dotdot
down
dyad
fat
font
from
fwd
gfont
gsize
hat
italic
lcol
left
lineup
lpile
mark
matrix
ndefine
over
pile
rcol
right
roman
rpile
size
sqrt
sub
sup
tdefine
tilde
to
under
up
vec
~, ^
{ }
"..."

## 7.23. Several Examples

Here is the complete source for several examples and for the three display equations in the introduction to this chapter.

### Square root

Input:

```
.EQ
x = {-b +- sqrt{b sup 2 - 4ac}} over 2a
.EN
```

Output:

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

### Summation, Integral, and Other Large Operators

Input:

```
.EQ
lim from {x -> pi /2} ( tan~x) = inf
.EN
```

Output:

$$\lim_{x\to\pi/2}(\tan x)=\infty$$

Input:

```
.EQ
sum from i=0 to infinity x sub i = pi over 2
.EN
```

Output:

$$\sum_{i=0}^{\infty}x_i=\frac{\pi}{2}$$

Input:

```
.EQ
lim from {x-> pi /2} ( tan~x) sup{sin~2x}~=~1
.EN
```

Output:

$$\lim_{x\to\pi/2}(\tan x)^{\sin 2x}=1$$

Input:

```
.EQ
define emx "{e sup mx}"
define mab "{m sqrt ab}"
define sa "{sqrt a}"
define sb "{sqrt b}"
int dx over {a emx - be sup -mx}~=~
left { lpile {
     1 over {2 mab} ~log~
          {sa emx - sb}over{sa emx + sb}
above
 1 over mab~tanh sup -1 ( sa over sb emx )
above
 -1 over mab~coth sup -1 ( sa over sb emx )
}
.EN
```

Output:

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

### Quoted Text

Input:

```
.EQ
lim~ roman "sup" ~x sub n = 0
.EN
```

Ouput:

$$\lim\ \mathrm{sup}\ x_n = 0$$

### Big Brackets

Input:

```
.EQ
left [ x+y over 2a right ]~=~1
.EN
```

Output:

$$\left[ \frac{x+y}{2a} \right] = 1$$

### Fractions

Input:

```
.EQ
a sub 0 + b sub 1 over
 {a sub 1 + b sub 2 over
  {a sub 2 + b sub 3 over
   {a sub 3 + ...}}}
.EN
```

Output:

$$a_0 + \frac{b_1}{a_1 + \frac{b_2}{a_2 + \frac{b_3}{a_3 + \cdots}}}$$

Input:

```
.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
```

Output:

$$G(z) = e^{\ln G(z)} = \exp\left(\sum_{k\ge1} \frac{S_k z^k}{k}\right) = \prod_{k\ge1} e^{S_k z^k / k}$$

Input:

```
.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
```

Output:

$$= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots\right)\left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \cdots\right) \cdots$$

Input:

```
.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 2} } over {2 sup k sub 2 k sub 2 ! } ~
...
{ S sub m sup {k sub m} } over {m sup k sub m k sub m ! }
right ) z sup m
.EN
```

Output:

$$=\sum_{m\geq0}\left[\sum_{\substack{k_1,k_2,\ldots,k_m\geq0\\k_1+2k_2+\cdots+mk_m=m}}\frac{S_1^{k_1}}{1^{k_1}k_1!}\;\frac{S_2^{k_2}}{2^{k_2}k_2!}\;\cdots\;\frac{S_m^{k_m}}{m^{k_m}k_m!}\right]z^m$$

### Shorthand for In-line Equations

Input:

```
.EQ
delim ##
.EN
```

```
Let #x sub i#, #y# and #alpha# be positive
```

Output:

Let $x_i$, $y$ and $\alpha$ be positive

# 8

# Verification Tools

# 8

# Verification Tools

## 8.1. spell

This command returns a list of misspelled words in a file. Because of the limited size of the on-line dictionary — less than 25,000 words — some words spell thinks are misspelled are in fact correct.

## 8.2. checknr

This program checks the syntax of troff files, in much the same way lint checks the syntax of C programs. People who try it often find it very helpful.

## 8.3. soelim

This program follows .so commands in troff files, incorporating the contents of these sourced files into the output. This program is helpful for searching groups of source files, and is also useful with preprocessors such as refer, tbl, and eqn, none of which follow source commands to fruition.

## 8.4. deroff

This command removes troff constructs from source files, and sends the results to standard output. Because some troff constructs necessarily contain text, some information may be lost from the output.

## 8.5. fmt

This command is a simplified formatter for use inside vi or mail. Devoid of hyphenation facilities, it does very little except fill text.

## 8.6. col

This command takes two-column text from nroff containing reverse line-feed escape sequences for the model 37 Teletype, and displays the two columns side-by-side, so they can be printed on a dumb lineprinter.

## 8.7. colcrt

This command is analogous to col, but was designed for CRT terminals, as it makes use of terminal capabilities when available.

## 8.8. ul

Also designed for CRTs, this command highlights underlined text using a terminal's underline mode, if available, and otherwise reverse video mode.

# Index

# Notes