Software Tools Text Formatter
User's Guide

Terrell L. Countryman
Perry B. Flinn
Jeanette T. Myers
Arnold D. Robbins
Peter N. Wan

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

July, 1984

TABLE OF CONTENTS

Foreword

'Fmt' is a program designed to facilitate the preparation of neatly formatted text. It provides many features, such as automatic margin alignment, paragraph indentation, hyphenation and pagination, that are designed to greatly ease an otherwise tedious job.

It is the intent of this guide to familiarize the user with the principles of automatic text formatting in general and with the capabilities and usage of 'fmt' in particular.

* Basics

Usage

'Fmt' takes as input a file containing text with interspersed formatting instructions. It is invoked by a command with various optional parameters, discussed below. The resultant output is appropriately formatted text suitable for a printer having backspacing capabilities. The output of 'fmt' is made available on its first standard output port, and so may be placed in a file, sent to a line printer, or changed in any of a number of ways, simply by applying standard Software Tools Subsystem I/O redirection.

When 'fmt' is invoked from the Subsystem, there are several optional parameters that may be specified to control its operation. The full command line syntax is

fmt [-s] [-p<first>[-<last>]] { <file name> }

A brief explanation of the cryptic notation: the items enclosed within square brackets ("[]") are optional -- they may or may not be specified; items enclosed between braces ("{}") may occur any number of times, including zero; items enclosed in angle brackets ("<>") designate character strings whose significance is suggested by the text within the brackets; everything else should be taken literally.

And now for an explanation of what these parameters mean:

-s — If this option is selected, 'fmt' will pause at the top of each page, ring the bell or buzzer on your terminal, and wait for a response. This feature is for the benefit of people using hard-copy terminals with paper not having pin-feed margins. The correct response, to be entered after the paper is mounted, is a control-c (hold the 'control' key down and type 'c').

-p ... — This option allows selection of which pages of the formatted document will actually be printed. Immediately following the "-p", without any intervening spaces, should be a number indicating the first page to be printed. Following this, a second number may be specified, separated from the first by a single dash, which indicates the last page to be printed. If this second number is omitted, all remaining pages will be produced.

<file> — Any number of file names may be specified on the command line. 'Fmt' will open the files in turn, formatting the contents of each one as if they constituted one big file. When the last named file is processed, 'fmt' terminates. If no file names are specified, standard input number one is used. In addition, standard input may be specified explicitly on

the command line by using a dash as a file name.

Commands and Text

'Fmt', like almost every other text formatter ever written, operates on an input stream that consists of a mixture of text and formatting commands. Each command starts at the beginning of a line with a 'control character', usually a period, followed by a two character name, in turn followed by some optional 'parameters'. There must not be anything else on the line. For example, in

```
.ta 11 21 31 41
```

the control character is a period, the command name is **ta**, and there are four parameters: "11", "21", "31" and "41". Notice that the command name and all the parameters must be separated from each other by one or more blanks. Anything not recognizable as a command is treated as text.

Filling and Margin Adjustment

Filled Text

'Fmt' collects as many words as will fit on a single output line before actually writing it out, regardless of line boundaries in its input stream. This is called 'filling' and is standard practice for 'fmt'. It can, however, be turned off with the 'no-fill' command

```
.nf
```

and lines thenceforth will be copied from input to output unaltered. When you want to turn filling back on again, you may do so with the 'fill' command

```
.fi
```

and 'fmt' will resume its normal behavior.

If there is a partially filled line that has not yet been written out when an **nf** command is encountered, the line is forced out before any other action is taken. This phenomenon of forcing out a partially filled line is known as a 'break' and occurs implicitly with many formatting commands. To cause one explicitly, the 'break' command

```
.br
```

is available.

Hyphenation

If, while filling an output line, it is discovered that the next word will not fit, an attempt is made to hyphenate it. Although 'fmt' is usually quite good in its choice of where to split a word, it occasionally makes a gaffe or two, giving reason to want to turn the feature off. Automatic hyphenation can be disabled with the 'no-hyphenation' command

.nh

long enough for a troublesome word to be processed, and then reenabled with the 'hyphenate' command

.hy

Neither command causes a break.

Margin Adjustment

After filling an output line, 'fmt' inserts extra blanks between words so that the last word on the line is flush with the right margin, giving the text a "professional" appearance. This is one of several margin adjustment modes that can be selected with the 'adjust' command

.ad <mode>

The optional parameter <mode> may be any one of four single characters: "b", "c", "l" or "r". If the parameter is "b" or missing, normal behavior will prevail -- both margins will be made even by inserting extra blanks between words. This is the default margin adjustment mode. If "c" is specified, lines will be shifted to the right so that they are centered between the left and right margins. If the parameter is "l", no adjustment will be performed; the line will start at the left margin and the right margin will be ragged. If "r" is specified, lines will be moved to the right so that the right margin is even, leaving the left margin ragged.

The 'no-adjustment' command

.na

has exactly the same effect as the following 'adjust' command:

.ad l

No adjustment will be performed, leaving the left margin even and the right margin ragged. In no case does a change in the adjustment mode cause a break.

Centering

Input lines may be centered, without filling, with the help of the 'center' command

```
.ce N
```

The optional parameter N is the number of subsequent input lines to be centered between the left and right margins. If the parameter is omitted, only the next line of input text is centered. Typically, one would specify a large number, say 1000, to avoid having to count lines; then, immediately following the lines to be centered, give a 'center' command with an parameter of zero. For example:

```
.ce 1000
more lines
than I care
to count
.ce 0
```

It is worth noting the difference between

```
.ce
```

and

```
.ad c
```

When the former is used, an implicit break occurs before each line is printed, preventing filling of the centered lines; when the latter is used, each line is filled with as many words as possible before centering takes place.

Sentence Punctuation

By default, 'fmt' adds an extra blank after punctuation at the end of a sentence; specifically, after periods, colons, exclamation points and question marks. This may not be desirable, particularly when abbreviations or a person's initials are involved. Thus, it can be turned on and off at will. The 'single-blank' command

```
.sb
```

turns the mode off, while the 'extra-blank' command

```
.xb
```

turns it back on again. As with hyphenation, neither command causes a break.

Summary - Filling and Margin Adjustment

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.ad <mode>	"b"	"b"	no	Set margin adjustment mode.
.br	-	-	yes	Force a break.
.ce N	N=0	N=1	yes	Center N input text lines.
.fi	on	-	no	Turn on fill mode.
.hy	on	-	no	Turn on automatic hyphenation.
.na	-	-	no	Turn off margin adjustment.
.nf	-	-	yes	Turn off fill mode. (Also inhibits adjustment.)
.nh	-	-	no	Turn off automatic hyphenation.
.sb	off	-	no	Single blank after end of sentence.
.xb	on	-	no	Extra blank after end of sentence.

Spacing and Page Control

Line Spacing

'Fmt' usually produces single-spaced output, but this can be changed, without a break, using the 'line-spacing' command

.ls N

The parameter N specifies how many lines on the page a single line of text will use; for double spacing, N would be two. If N is omitted, the default (single) spacing is reinstated.

Blank lines may be produced with the 'space' command

.sp N

The parameter N is the number of blank lines to produce; if omitted, a value of one is assumed. The **sp** command first causes a break; this not only causes a partially filled line to be output, but if the current line spacing is more than one, the break will cause the extra blank lines to be output as well. Then the blank lines generated by **sp** are output. Thus, if output is being double-spaced and the command

.sp 3

is given, four blank lines will be generated: one from the double-spacing that is in effect, and three from the **sp** command. If the value of N calls for more blank lines than there are remaining on the current page, any extra ones are discarded. This ensures that, normally, each page begins at the same distance from the top of the paper.

Page Division

'Fmt' automatically divides its output into pages, leaving adequate room at the top and bottom of each page for running headings and footings. There are several commands that facilitate the control of page divisions when the normal behavior is inadequate.

The 'begin-page' command

.bp ±N

causes a break and a skip to the top of the next page. If a parameter is given, it serves to alter the page number and so it must be numeric with an optional plus or minus sign. If the parameter is omitted, the page number is incremented by one. If the command occurs at the top of a page before any text has been printed on it, the command is ignored, except perhaps to set the page number. This is to prevent the random occurrence of blank pages.

The optionally signed numeric parameter is a form of parameter used by many formatting commands. When the sign is omitted, it indicates an absolute value to be used; when the sign is present, it indicates an amount to be added to or subtracted from the current value.

The page number may be set independently of the 'begin-page' command with the 'page-number' command

.pn ±N

The next page after the current one, when and if it occurs, will be numbered ±N. No break is caused.

The length of each page produced by 'fmt' is normally 66 lines. This is standard for eleven inch paper printed at six lines per inch. However, if non-standard paper is used, the

printed length of the page may easily be changed with the 'page-length' command

.pl ±N

which will set the length of the page to ±N lines without causing a break.

It is possible skip an arbitrary number of pages in a controlled fashion. To do this, use the 'page-skip' command

.ps <max> <modulus>

<Max> is the maximum number of pages plus one that 'fmt' will skip. <Modulus> is the number which 'fmt' uses modulo the next output page number to count skipping pages. It works as follows: 'Fmt' sees the **.ps** command. It computes the page number of the current page plus one, and then takes the remainder of that number divided by the <modulus>, and saves it. 'Fmt' skips pages, adding one to this saved value. As long as this value is less than <max>, it continues to skip pages. For instance, if the current page is 15, and you issue a

.ps 3 5

command, 'fmt' would compute ((15 + 1) mod 5), yielding (16 mod 5), which is one (16 divided by 5 is 3, with 1 left over). It will then skip two pages, since it started with one, then skipped one, which is two. This is still less than three, so it skips one more page, yielding three, which is not less than three, so it stops. It is really quite simple. For instance, to skip to the next even page, use

.ps 2 2

and to skip to the next odd page, use

.ps 1 2

This feature is particularly useful for writing macros which aid with large documents. For example, it may be necessary that a chapter always start on an odd numbered page. So the 'begin chapter' macro would have a '**.ps** 1 2' as one of its lines. (See later for more details on how to write macros.)

Finally, if it is necessary to be sure of having enough room on a page, say for a figure or a graph, use the 'need' command

.ne N

'Fmt' will cause a break, check if there are N lines left on the current page and, if so, will do nothing more. Otherwise, it will skip to the top of the next page where there should be adequate room.

'No-space' Mode

'No-space' mode is a feature that assists in preventing unwanted blank lines from appearing, usually at the top of a page. When in effect, certain commands that cause blank lines to be generated, such as **bp**, **ne** and **sp**, are suppressed. For the most part, 'no-space' mode is managed automatically; it is turned on automatically at the top of each page before the first text has appeared, and turned off again automatically when a line of output is generated. This accounts for the suppression of **bp** commands at the top of a page and the discarding of excess blank lines in **sp** commands.

'No-space' mode may be turned on explicitly with the 'no-space' command

.ns

and turned off explicitly with the 'restore-spacing' command

.rs

Neither command causes a break.

Summary - Spacing and Page Control

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.bp ±N	N=1	next	yes	Begin a new page.
.ls N	N=1	N=1	yes	Set line spacing.
.ne N	-	N=1	yes	Express a need for N contiguous lines.
.ns	on	-	no	Turn on 'no-space' mode.
.pl ±N	N=66	N=66	no	Set page length.
.pn ±N	N=1	ignored	no	Set page number.
.ps N M	N=M=0	N=M=0	yes	Skip pages while (page number mod M) is less than N.
.rs	-	-	no	Turn off 'no-space' mode.
.sp N	-	N=1	yes	Put out N blank lines.

Margins and Indentation

Margins

All formatting operations are performed within the framework of a page whose size is defined by four margins: top, bottom, left and right. The top and bottom margins determine the number of lines that are left blank at the top and bottom of each page. Likewise, the left and right margins determine the first and last columns across the page into which text may be placed.

Top and Bottom Margins

Both the top and the bottom margins consist of two sub-margins that fix the location of the header and footer lines. For the sake of clarity, the first and second sub-margins of the top margin will be referred to as 'margin 1' and 'margin 2', and the first and second sub-margins of the bottom margin, 'margin 3' and 'margin 4'.

The value of margin 1 is the number of lines to skip at the top of each page before the header line, plus one. Thus, margin 1 includes the header line and all the blank lines preceding it from the top of the paper. By default, its value is three. Margin 2 is the number of blank lines that are to appear between the header line and the first text on the page. Normally, it has a value of two. The two together form a standard top margin of five lines, with the header line right in the middle. It is easy enough to change these defaults if they prove unsatisfactory; just use the 'margin-1' and 'margin-2' commands

```
.m1 ±N
.m2 ±N
```

to set either or both sub-margins to ±N.

The bottom margin is completely analogous to the top margin, with margin 3 being the number of blank lines between the last text on a page and the footer line, and margin 4 being the number of lines from the footer to the bottom of the paper (including the footer). They may be set using the 'margin-3' and 'margin-4' commands

```
.m3 ±N
.m4 ±N
```

which work just like their counterparts in the top margin; none causes a break.

Left and Right Margins

The left and right margins define the first and last columns into which text may be printed. They affect such things as

adjustment and centering. The left margin is normally set at column one, though this is easily changed with the 'left-margin' command

.lm ±N

The right margin, which is normally positioned in column sixty, can be set similarly with the 'right-margin' command

.rm ±N

To ensure that the new margins apply only to subsequent text, each command causes a break before changing the margin value.

Indentation

It is often desirable to change the effective value of the left margin for indentation, without actually changing the margin itself. For instance, all of the examples in this guide are indented from the left margin in order to set them apart from the rest of the text. Indentation is easily arranged using the 'indent' command,

.in ±N

whose parameter specifies the number of columns to indent from the left margin. The initial indentation value, and the one assumed if no parameter is given, is zero (i.e., start in the left margin).

For the purpose of margin adjustment, the current indentation value is added to the left margin value to obtain the effective left margin. In this respect, the **lm** and **in** commands are quite similar. But, whereas the left margin value affects the placement of centered lines produced by the **ce** command, indentation is completely ignored when lines are centered.

Paragraph indentation poses a sticky problem in that the indentation must be applied only to the first line of the paragraph, and then normal margins must be resumed. This can't be done conveniently with the 'indent' command, since it causes a break. Therefore, 'fmt' has a 'temporary-indent' command

.ti ±N

whose function is to cause a break, alter the current indentation value by ±N until the next line of text is produced, and then reset the indentation to its previous value. So to begin a new paragraph with a five column indentation, one would say

.ti +5

Page Offset

As if control over the left margin position and indentation were not enough, there is yet a third means for controlling the position of text on the page. The concept of a page offset involves nothing more than prepending a number of blanks to each and every line of output. It is primarily intended to allow output to be easily positioned on the paper without having to adjust margins and indentation (with all their attendant side effects) and without having to physically move the paper. Although the page offset is initially zero, other arrangements may be made with the 'page-offset' command

.po ±N

which causes a break.

'Eo' and 'oo' commands allow you to specify different page offsets for even- and odd-numbered pages respectively. Like 'po', they are initialized to zero and revert to that value when no parameter is specified. For instance,

.eo ±N

will change the even-numbered page offset by N (or to N if no sign is specified).

Margin Characters

It is common practice in the revision of technical literature to indicate parts of the text that are different from previous versions of the same document. Such changes are usually indicated by "revision bars" which are vertical lines in the left margin of lines that are new or revised. 'Fmt' provides for this capability with two formatting commands. The 'margin-offset' command,

.mo ±N

without causing a break, specifies that ±N columns are to be reserved between the 'page-offset' columns and the 'left-margin' column for revision bars or other marginal characters. The margin offset starts out at zero, and reverts to that value if no parameter is specified.

Once a non-zero margin offset has been set, any arbitrary character may be placed in the leftmost column of the area with the 'margin-character' command:

.mc <char>

Initially, and when <char> is omitted, this character has blank as its value. For revision bars, <char> would be specified as "|". Whatever character is specified, it is placed next to the left margin on every line of output as long as the margin offset

is non-zero.

Summary - Margins and Indentation

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.eo ±N	N=0	N=0	yes	Set even page offset.
.in ±N	N=0	N=0	yes	Indent left margin.
.lm ±N	N=1	N=1	yes	Set left margin.
.m1 ±N	N=3	N=3	no	Set top margin before and including page heading.
.m2 ±N	N=2	N=2	no	Set top margin after page heading.
.m3 ±N	N=2	N=2	no	Set bottom margin before page footing.
.m4 ±N	N=3	N=3	no	Set bottom margin including and after page footing.
.mc <char>	BLANK	BLANK	no	Set margin character.
.mo ±N	N=0	N=0	no	Set margin offset.
.oo ±N	N=0	N=0	yes	Set odd page offset.
.po ±N	N=0	N=0	yes	Set page offset.
.rm ±N	N=60	N=60	yes	Set right margin.
.ti ±N	N=0	N=0	yes	Temporarily indent left margin.

Headings, Footings and Titles

Three Part Titles

A three part title is a line of output consisting of three segments. The first segment is left-justified, the second is centered between the left and right margins, and the third is right-justified. For example

left part center part right part

is a three part title whose first segment is "left part", whose second segment is "center part", and whose third segment is "right part".

To generate a title at the current position on the page, the 'title' command is available:

.tl /left part/center part/right part/

In fact, this command was used to generate the previous example. The parameter to the title command is made up of the text of the three parts, with each segment enclosed within a pair of delimiter characters. Here, the delimiter is a slash, but any other character may be used as long as it is used consistently within the same command. If one or more segments are to be omitted, indicate this with two adjacent delimiters at the desired position. Thus,

.tl ///Page 1/

specifies only the third segment and would produce something like this:

Page 1

It is not necessary to include the trailing delimiters.

To facilitate page numbering, you may include the sharp character ("#") anywhere in the text of the title; when the command is actually performed, 'fmt' will replace all occurrences of the "#" with the current page number. To produce a literal sharp character in the title, it should be preceded by an "@"

@#

so that it loses its special meaning.

The first segment of a title always starts at the left margin as specified by the **lm** command. While the third segment normally ends at the right margin as specified by the **rm** command, this can be changed with the 'length-of-title' command:

.lt ±N

which changes the length of subsequent titles to ±N, still beginning at the left margin. Note that the title length is automatically set by the **lm** and **rm** commands to coincide with the distance between the left and right margins.

Page Headings and Footings

The most common uses for three part titles are page headings and footings. The header and footer lines are initially blank. Either one or both may be set at any time, without a break, by using the 'header' command

```
.he /left/center/right/
```

to set the page heading, and the 'footer' command

```
.fo /left/center/right/
```

to set the page footing. The change will become manifest the next time the top or the bottom of a page is reached. As with the **tl** command, the "#" may be used to access the current page number.

It is often desirable when producing text to be printed on both sides of a page to have different headings and footings on odd- and even-numbered pages. Although the **he** and **fo** commands affect the headings and footings on all pages, it is possible to set up independent headings and footings for odd- and even-numbered pages. For odd-numbered pages, the 'odd-header' and 'odd-footer' commands are available:

```
.oh /left/center/right/
.of /left/center/right/
```

while the 'even-header' and 'even-footer' commands are provided for even-numbered pages:

```
.eh /left/center/right/
.ef /left/center/right/
```

As an illustration, the following commands were used to generate the page headings and footings for this guide:

```
.eh /Text Formatter User's Guide///
.oh ///Text Formatter User's Guide/
.fo //- # -//
```

Summary - Headings, Footings and Titles

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.ef /l/c/r/	blank	blank	no	Set even-numbered page footing.
.eh /l/c/r/	blank	blank	no	Set even-numbered page heading.
.fo /l/c/r/	blank	blank	no	Set running page footing.
.he /l/c/r/	blank	blank	no	Set running page heading.
.lt ±N	N=60	N=60	no	Set length of header, footer and titles.
.of /l/c/r/	blank	blank	no	Set odd-numbered page footing.
.oh /l/c/r/	blank	blank	no	Set odd-numbered page heading.
.tl /l/c/r/	blank	blank	yes	Generate a three part title.

Tabulation

Tabs

Just like any good typewriter, 'fmt' has facilities for tabulation. When it encounters a special character in its input called the 'tab character' (analogous to the TAB key on a typewriter), it automatically advances to the next output column in which a 'tab stop' has been previously set. Tab stops are always measured from the effective left margin, that is, the left margin plus the current indentation or temporary indentation value. Whatever column the left margin may actually be in, it is always assumed to be column one for the purpose of tabulation.

Originally, a tab stop is set in every eighth column, starting with column nine. This may be changed using the 'tab' command

.ta <col> <col> ...

Each parameter specified must be a number, and causes a tab stop to be set in the corresponding output column. All existing stops are cleared before setting the new ones, and a stop is set in every column beyond the last one specified. This means that if

no columns are specified, a stop is set in every column.

By default, 'fmt' recognizes the ASCII TAB, control-i, as the 'tab character'. But since this is an invisible character and is guaranteed to be interpreted differently by different terminals, it can be changed to any character with the 'tab-character' command:

.tc <char>

While there is no restriction on what particular character is specified for <char>, it is wise to choose one that doesn't occur too frequently elsewhere in the text. If you omit the parameter, the tab character reverts to the default.

When 'fmt' expands a tab character, it normally puts out enough blanks to get to the next tab stop. In other words, the default 'replacement' character is the blank. This too may easily be changed with the 'replacement-character' command:

.rc <char>

As with the **tc** command, <char> may be any single character. If omitted, the default is used.

A common alternate replacement character is the period, which is frequently used in tables of contents. The following example illustrates how one might be constructed:

```
.ta 52
.tc \
Section Name\Page
.rc .
.sp
.nf
.ta 53
Basics\1
Filling and Margin Adjustment\2
Spacing and Page Control\5
.sp
.fi
```

The result should look about like this:

```
Section Name                                        Page

Basics.....................................................1
Filling and Margin Adjustment..............................2
Spacing and Page Control...................................5
```

A final word on tabs: Since the default replacement character is a blank you might think that, in the process of adjusting margins (i.e., when the adjustment mode is "b"), 'fmt' might throw in extra blanks between words that were separated by the tab character. Since this is definitely <u>not</u> the expected or

desired behavior, 'fmt' uses what is called a "phantom blank" as the default replacement character. The phantom blank prints as an ordinary blank, but is not recognized as one during margin adjustment.

Summary - Tabulation

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.ta N ...	9 17 ...	all	no	Set tab stops.
.tc c	TAB	TAB	no	Set tab character.
.rc c	BLANK	BLANK	no	Set tab replacement character.

Miscellaneous Commands

Comments

It is rare that a document survives its writing under the pen of just one author or editor. More frequently, several different people are likely to put in their two cents worth concerning its format or content. So, if the author is particularly attached to something he has written, he is well advised to say so. Comments are an ideal vehicle for this purpose and are easily introduced with the 'comment' command

.# <commentary text>

Everything after the **#** up to and including the next newline character is completely ignored by 'fmt'.

Boldfacing and Underlining

'Fmt' makes provisions for **boldfacing** and <u>underlining</u> lines or parts thereof with two commands:

.bf N

boldfaces the next N lines of input text, while

.ul N

underlines the next N lines of input text. In both cases, if N is omitted, a value of one is assumed. Neither command causes a break, allowing single words or phrases to be boldfaced or underlined without affecting the rest of the output line.

It is also possible to use the two in combination. For instance, the heading at the beginning of the table of contents was produced by a sequence of commands and text similar to the following:

```
.bf
.ul
TABLE OF CONTENTS
```

As with the 'center' command, these two commands are often used to bracket the lines to be affected by specifying a huge parameter value with the first occurrence of the command and a value of zero with the second:

```
.bf 1000
.ul 1000
lots of lines
to be
boldfaced
and
underlined
.bf 0
.ul 0
```

Control Characters

As mentioned in the first section, command lines are distinguished from text by the presence of a 'control character' in column one. In all the examples cited thus far, a period has been used to represent the control character. It is possible to select any character for this purpose. In fact, several occasions arose in the writing of this guide which called for use of an alternate control character, particularly in the construction of the command summaries at the end of each section. The 'control-character' command may be used anywhere to select a new value:

```
.cc <char>
```

The parameter <char>, which may be any single character, becomes the new control character. If the parameter is omitted, the familiar period is reinstated.

It has been shown that many commands automatically cause a break before they perform their function. When this presents a problem, it can be altered. If instead of using the basic control character the 'no-break' control character is used to introduce a command, the automatic break that would normally result is suppressed. The standard no-break control character is the grave accent ("`"), but may easily be changed with the following command:

```
.c2 <char>
```

As with the **cc** command, the parameter may be any single character, or may be omitted if the default value is desired.

Prompting

Brief, one-line messages may be written directly to the user's terminal using the 'prompt' command

.er <brief, one-line message>

The text that is actually written to the terminal starts with the first non-blank character following the command name, and continues up to, but not including, the next newline character. If a newline character should be included in the message, the escape sequence

@n

may be used. Leading blanks may also be included in the message by preceding the message with a quote or an apostrophe. 'Fmt' will discard this character, but will then print the rest of the message verbatim. For instance,

.er ' this is a message with 10 leading blanks

would write the following text on the terminal, leaving the cursor or carriage at the end of the message

this is a message with 10 leading blanks

For a multiple-line message, try

.er multiple@nline@nmessage@n

The output should look like this:

multiple
line
message

Prompts are particularly useful in form letter applications where there may be several pieces of information that 'fmt' has to ask for in the course of its work. The next section describes how 'fmt' can dynamically obtain information from the user.

Premature Termination

If 'fmt' should ever encounter an 'exit' command

.ex

in the course of doing its job, it will cause a break and exit immediately to the Subsystem.

Summary - Miscellaneous Commands

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.#	-	-	no	Introduce a comment.
.bf N	N=0	N=1	no	Boldface N input text lines.
.c2 c	`	`	no	Set no-break control character.
.cc c	.	.	no	Set basic control character.
.er text	-	ignored	no	Write a message to the terminal.
.ex	-	-	yes	Exit immediately to the Subsystem.
.ul N	N=0	N=1	no	Underline N input text lines.

Input/Output Processing

Input File Control

Up to this point, it has been assumed that 'fmt' reads only from its standard input file or from files specified as parameters on the command line. It is also possible to dynamically include the contents of any file in the midst of formatting another. This aids greatly in the modularization of large, otherwise unwieldy documents, or in the definition of frequently used sequences of commands and text.

The 'source' command is available to dynamically include the contents of a file:

.so <file>

The parameter <file> is mandatory; it may be an arbitrary file system pathname, or, as with file names on the command line, a single dash ("-") to specify standard input number one.

The effect of a 'source' command is to temporarily preempt the current input source and begin reading from the named file. When the end of that file is reached, the original source of input is resumed. Files included with 'source' commands may themselves contain other 'source' commands; in fact, this

'nesting' of input files may be carried out to virtually any depth.

'Fmt' provides one additional command for manipulating input files. The 'next file' command

.nx <file>

may be used for either one of two purposes. If you specify a <file> parameter, all current input files are closed (including those opened with **so** commands), and the named file becomes the new input source. You can use this for repeatedly processing the same file, as, for example, with a form letter. If you omit the <file> parameter, 'fmt' still closes all of its current input files. But instead of using a file name you supply with the **nx** command, it uses the next file named on the command line that invoked 'fmt'. If there is no next file, then formatting terminates normally.

Neither the **so** command nor the **nx** command causes a break.

Output File Control

The output of the formatter is always written on STDOUT unless you divert it with the divert output stream command, 'dv'. 'Dv' can be used to divert fmt's output to a named file:

.dv <file>

All output is written in <file> until a 'dv' command with no parameter is specified. 'Dv' can also be used to divert output to a temporary file that can be later read with the 'so' command. This is useful for generating tables of contents for documents (see the "Application Notes" section). The command

.dv N

diverts output to stream 'N' and can be read at any time and repeatedly by the command

.so N

Output will be diverted until the the 'dv' command is seen again without parameters. (N can be an integer between 1 and 100; the upper limit may be somewhat less for you --- it depends on the number of file units that you can have open and the number of file units that you actually have open at the time the command is executed).

The basic difference between the two variants of 'dv' is that 'dv <file>' opens <file> for WRITE access; <file> cannot be used as an input file. 'Dv N' opens a temporary file for READ/WRITE access; therefore, 'so N' causes the temporary file to be rewound and read. If the command 'dv N' occurs a second, third, fourth etc. time, diverted lines are appended to the end

of that same temporary file.

One final important comment is necessary. We were hesitant to even tell you about 'dv' because of its rather nasty habit of executing commands instead of diverting them. Since it is the only way for you to generate automatic table of contents we decided to document it. Just keep in mind that when you want to divert commands, precede them by a character other than your control character; you can later designate that character as your control character before you read the stream.

Functions, Variables and Special Characters

Whenever 'fmt' reads a line of input, no matter what the source may be, there is a certain amount of 'pre-processing' done before any other formatting operations take place. This pre-processing consists of the interpretation of 'functions', 'variables' and 'special characters'. A 'function' is a predefined set of actions that produces a textual result, possibly based on some user supplied textual input. For example, one hypothetical function might be named 'time', and its result might be a textual representation of the current time of day:

```
01:22:49
```

A 'variable' is simply one of 'fmt's internal parameters, such as the current page length or the current line-spacing value; the name of each variable is the same as the two-character name of the corresponding command to set the value of that parameter. The result of a variable is just a textual representation of that value.

A 'special character' is like a function or variable, but its result is a single character that cannot be conveniently generated from the keyboard.

From the standpoint of a user, functions, variables and special characters are all very similar. In fact, they are invoked identically by enclosing the appropriate name, plus any text to be used as arguments, in square brackets:

```
[bf This text to be boldfaced]
[ls]
[alpha 5]
```

Such a construct is known as a "function call."

When 'fmt' sees a function call in an input line, it excises everything in between the brackets, including the brackets themselves, and inserts the results in its place. Naturally, anything not recognizable is left alone. If by chance you want the name of a function, variable or special character enclosed in square brackets included literally as part of the text, you can inhibit evaluation by preceding the left bracket with the escape character:

```
@[time]
```

Similarly, a right bracket may appear literally inside a function call when preceded by an escape character:

```
[bf [item 1@]]
```

It is also possible to "nest" function calls so that the results of one may be used as arguments to another:

```
[bf [ldate]]
```

Number Registers

The 'number registers' are a group of 200 accumulators (numbered 1-200) on which simple arithmetic operations may be performed. They find their greatest use in the preparation of documents with numbered sections and paragraphs. Number registers are accessed and manipulated by a special set of functions. The 'set' function

```
[set reg value]
```

assigns the integer 'value' to the register 'reg' and yields the empty string as its result. The 'add' function

```
[add reg value]
```

adds the integer 'value' (which, by the way may be positive or negative) to the register 'reg'. This function too yields an empty result. Finally, the 'num' function

```
[num reg]
```

yields the current value of the register 'reg' as its result. In addition, 'reg' may either be prefixed or postfixed by a plus or minus sign. If the sign appears before the register number, the register is incremented or decremented (according to the sign) by one before the function's result is yielded. If the sign follows the register number, though, the register's current value is yielded and then the register is incremented or decremented.

Functions

The following table summarizes the available functions:

add	Add constant to number register
bf	Boldface the arguments on output
cu	Output the arguments with a continuous underline
date	Current date; e.g., 11/27/84
day	Current day of the week; e.g., Tuesday
ldate	Current date: e.g., November 27, 1984
num	Output value of number register with optional pre-

	or post-incrementation or decrementation
rn	Convert the argument to a lower-case Roman numeral
RN	Convert the argument to an upper-case Roman numeral
set	Set number register to value
sub	Output the arguments as a subscript (requires post-processor, e.g. 'sprint')
sup	Output the arguments as a superscript (requires post-processor)
time	Current time of day; e.g., 01:22:54
ul	Underline the arguments on output
letter	Convert a number to its lower case equivalent
LETTER	Convert a number to its upper case equivalent
vertspace	Change the vertical spacing on a NEC Spinwriter (requires spinwriter)
even	Test if number is even
odd	Test if number is odd
cap	Capitalize Text
small	Map all characters of text to lower case
plus	Add two numbers
minus	Subtract two numbers
header	Return the page header
evenheader	Return the even page header
oddheader	Return the odd page header
footer	Return the page footer
evenfooter	Return the even page footer
oddfooter	Return the odd page footer
cmp	Perform string comparison
icmp	Perform integer comparison
bottom	Return the number of the last printed line
top	Return the number of the first printed line

Variables

The formatting parameters whose values are available through function calls are summarized in the following table:

cc	Current basic control character
c2	Current no-break control character
in	Current indentation value
lm	Current left margin value
ln	Current line number on the page
ls	Current line-spacing value
lt	Length of titles
ml	Current macro invocation level
m1	Current margin 1 value
m2	Current margin 2 value
m3	Current margin 3 value
m4	Current margin 4 value
ns	True or false if no-space is in effect
pl	Current page length value
pn	Current page number
po	Current page offset value
rm	Current right margin value
tc	Current tab character

	ti	Current temporary indentation value
	tcpn	Current page number, right justified in 4 character field

Special Characters

The following table summarizes the available special characters. In each case, a positive integer may be included as an argument following the name to produce multiple instances of the character. For example, "[bl 5]" yields five contiguous phantom blanks. NOTE: in order for the Greek letters and mathematical symbols to be printed correctly, a post-processor such as 'dprint' (see Section 3 of the Software Tools Subsystem Reference Manual) and/or special printing equipment is required.

	bl	Phantom blank
	bs	Backspace
	alpha	lower-case Greek alpha
*	ALPHA	upper-case Greek alpha
	beta	lower-case Greek beta
*	BETA	upper-case Greek beta
*	chi	lower-case Greek chi
*	CHI	upper-case Greek chi
	delta	lower-case Greek delta
*	DELTA	upper-case Greek delta
	epsilon	lower-case Greek epsilon
*	EPSILON	upper-case Greek epsilon
	eta	lower-case Greek eta
*	ETA	upper-case Greek eta
	gamma	lower-case Greek gamma
	GAMMA	upper-case Greek gamma
	infinity	infinity symbol
	integral	integration symbol
*	INTEGRAL	large integration sign
*	iota	lower-case Greek iota
*	IOTA	upper-case Greek iota
*	kappa	lower-case Greek kappa
*	KAPPA	upper-case Greek kappa
	lambda	lower-case Greek lambda
	LAMBDA	upper-case Greek lambda
	mu	lower-case Greek mu
*	MU	upper-case Greek mu
	nabla	inverted delta (APL del)
	not	EBCDIC-style not symbol
*	nu	lower-case Greek nu
*	NU	upper-case Greek nu
	omega	lower-case Greek omega
	OMEGA	upper-case Greek omega
*	omicron	lower-case Greek omicron
*	OMICRON	upper-case Greek omicron
	partial	partial differential symbol
	phi	lower-case Greek phi
	PHI	upper-case Greek phi
	psi	lower-case Greek psi
	PSI	upper-case Greek psi

	pi	lower-case Greek pi
	PI	upper-case Greek pi
	rho	lower-case Greek rho
*	RHO	upper-case Greek rho
	sigma	lower-case Greek sigma
	SIGMA	upper-case Greek sigma
	tau	lower-case Greek tau
*	TAU	upper-case Greek tau
	theta	lower-case Greek theta
	THETA	upper-case Greek theta
*	upsilon	lower-case Greek upsilon
*	UPSILON	upper-case Greek upsilon
	xi	lower-case Greek xi
*	XI	upper-case Greek xi
	zeta	lower-case Greek zeta
*	ZETA	upper-case Greek zeta
*	downarrow	arrow pointing down
*	uparrow	arrow pointing up
*	backslash	back slash symbol
*	tilde	tilde symbol
*	largerbrace	large square right brace
*	largelbrace	large square left brace
*	proportional	proportional symbol
*	apeq	approximately equal to
*	ge	greater than or equal to
*	imp	implies
*	exist	there exists
*	AND	logical and
*	ne	not equal to
*	psset	proper subset
*	sset	subset
*	le	less than or equal to
*	nexist	there does not exist
*	univ	for every
*	OR	logical or
*	iso	congruence
*	lfloor	left floor
*	rfloor	right floor
*	lceil	left ceiling
*	rceil	right ceiling
*	small0	a small 0
*	small1	a small 1
*	small2	a small 2
*	small3	a small 3
*	small4	a small 4
*	small5	a small 5
*	small6	a small 6
*	small7	a small 7
*	small8	a small 8
*	small9	a small 9
*	scolon	semicolon
*	dquote	double quote
*	dollar	dollar sign

The special characters marked with an asterisk (*) are only available on the NEC **Spinwriter,** and so the output of 'fmt' must

be post-processed with 'sprint'.

In particular, these characters require that the special Times-Roman/Mathematics type wheel be in the **Spinwriter**. This wheel, in order to accommodate the special characters, lacks certain of the regular ASCII graphics. These are substituted for by special functions. For example, [scolon] is used to produce a semi-colon.

Summary - Input Processing

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.dv [stream]	-	end '.dv'	no	Temporarily divert the output stream to a "filename" or to a temporary file designated by an integer "N" (to be later read by a ".so N" command) until a 'dv' command with no arguments is seen.
.nx file	-	next arg	no	Move on to the next input file.
.so <stream>	-	ignored	no	Temporarily alter the input source. "Stream can be a "-" to indicate standard input, a "filename," or an integer "N" corresponding to a temporary file created by a previous '.dv N' command.

Macros

Macro Definition

A macro is nothing more than a frequently used sequence of commands and/or text that have been grouped together under a single name. This name may then be used just like an ordinary command to invoke the whole group in one fell swoop.

The definition (or redefinition) of a macro starts with a 'define' command

```
.de xx
```

whose parameter is a one or two character string that becomes the name of the macro. The macro name may consist of any characters other than blanks, tabs or newlines; upper and lower letters are distinct. The definition of the macro continues until a matching 'end' command

```
.en xx
```

is encountered. Anything may appear within a macro definition, including other macro definitions. The only processing that is done during definition is the interpretation of variables and functions (i.e. things surrounded by square brackets). Other than this, lines are stored exactly as they are read from the input source. To include a function call in the definition of a macro so that its interpretation will be delayed until the macro is invoked, the opening bracket should be preceded by the escape character "@". For example,

```
.# tm --- time of day
.de tm
@[time]
.en tm
```

would produce the current time of day when invoked, whereas

```
.# tm --- time of day
.de tm
[time]
.en tm
```

would produce the time at which the macro definition was processed.

Macro Invocation

Again, a macro is invoked like an ordinary command: a control character at the beginning of the line immediately followed by the name of the macro. So to invoke the above 'time-of-day' macro, one might say

```
.tm
```

As with ordinary commands, macros may have parameters. In fact, anything typed on the line after the macro name is available to the contents of the macro. As usual, blanks and tabs serve to separate parameters from each other and from the macro name. If it is necessary to include a blank or a tab within a parameter, it may be enclosed in quotes. Thus,

```
"parameter one"
```

would constitute a single parameter and would be passed to the

macro as

```
parameter one
```

To include an actual quotation mark within the parameter, type two quotes immediately adjacent to each other. For instance,

```
"""quoted string"""
```

would be passed to the macro as the single parameter

```
"quoted string"
```

Within the macro, parameters are accessed in a way similar to functions and variables: the number of the desired parameter is enclosed is square brackets. Thus,

```
[1]
```

would retrieve the first parameter,

```
[2]
```

would fetch the second, and so on. As a special case, the name of the macro itself may be accessed with

```
[0]
```

Assume there is a macro named "mx" defined as follows:

```
.# mx --- macro example
.de mx
Macro named '[0]', invoked with two arguments:
'[1]' and '[2]'.
.en mx
```

Then, typing

```
.mx "param 1" "param 2"
```

would produce the same result as typing

```
Macro named 'mx', invoked with two arguments:
'param 1' and 'param 2'.
```

Macros are quite handy for such common operations as starting a new paragraph, or for such tedious tasks as the construction of tables like the ones appearing at the end of each section in this guide. For some examples of frequently used macros, see the applications notes in the following pages.

Appending To A Macro

It is possible to add text to the body of a previously defined macro, using the 'append macro' command:

.am xx

where xx is a previously defined macro. It is an error to append to a macro which has not been previously defined. The additional text of the macro is terminated with a '.en xx' command, just like the initial definition of the macro. The rules for the additional text of the macro are the same as for the initial text, i.e. any function calls or special characters must be escaped with an "@" sign to prevent their immediate evaluation.

Summary - Macros

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.de xx	-	ignored	no	Begin definition or redefinition of a macro.
.en xx	-	ignored	no	End macro definition.
.am xx	-	ignored	no	Add additional text to the body of a previously defined macro.

Conditional Line Processing

Introduction

This sections discusses the features of 'fmt' which provide you with considerable control and flexibility over the formatting of your documents.

The .if command

'Fmt' allows you to test a condition and if that condition is true, it will execute a command. Optionally, you may specify a command to be executed if the condition is not true (an 'else' part). This is done using the 'if' command:

.if cond delim true_part [delim else_part]

This evaluates a condition ('cond') which, if it is true, will cause 'true_part' to be executed, just as if 'true_part' had been on a line by itself. If the condition is false, and the

'else_part' is present, then 'else_part' will be executed as if it had been on a line by itself. The 'delim' is any single non-blank character. For instance, the command

.if [odd [pn]] / .er odd page@n / .er even page@n

will write either 'odd page' or 'even page' to the terminal, depending on whether or not the current page is odd (the [odd] function will be discussed shortly).

The 'cond' can be negated by putting a '~' in front of it. Note that 'fmt' only checks for a single '~' to see if the condition is to be inverted. 'Fmt' is not a true programming language! It is probably almost always better to rewrite your condition than to use a '~' to negate it. The functions discussed below, and the ability to specify an 'else' part, provide ample flexibility to do whatever needs to be done.

A **.if** command with no arguments has no effect on the formatted output. The **.if** command may or may not cause a break, depending on the contents of the 'if' and 'else' parts.

Conditional Functions

'Fmt' provides four function calls which return either true or false (1 or 0) depending on the truth values of the conditions specified in their arguments. The four functions are as follows:

odd — Return true (false) if its integer argument is odd (even).

even — Return true (false) if its integer argument is even (odd).

cmp — Does a string comparison on its arguments, returning true if the specified relation is true, false otherwise. The form of this call is described below.

icmp — Does an integer comparison on its arguments, returning true if the specified relation is true, false otherwise.

The two comparison functions are called with three arguments, the first operand, a relational operator, and the second operand. The relational operators are:

<=	Less than or equal to.
=<	Less than or equal to.
<	Less than.
==	Equal to.
=	Equal to.
~=	Not equal to.

```
      <>        Not equal to.
      ><        Not equal to.
      >=        Greater than or equal to.
      =>        Greater than or equal to.
      >         Greater than.
```

A missing or incorrect operator is an error, and will cause 'fmt' to exit. As an example, to determine where you are, you could do the following:

```
            This must be
            .if [cmp [day] = Tuesday] / Belgium. /  Somewhere.
```

would cause the output to be "This must be Belgium." if it were Tuesday. Otherwise your text would simply wonder where it is.

Summary - Conditional Line Processing

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.if <args>	-	ignored	maybe	Conditional execution of an input line.

Applications Notes

This section will illustrate the capabilities of 'fmt' with some actual applications. Most of the examples are macros that assist in common formatting operations, but attention has also been given to table construction. All of the macros presented here are available for general use in the file "//extra/fmacro/report", which may be named on the command line invoking 'fmt' or may be included with a 'source' command as follows:

```
.so =fmac=/report
```

Paragraphs

One standard way of beginning a new paragraph is to skip a line and indent by a few spaces, as was done throughout this guide. This can be done by giving an **sp** command followed by a **ti** command. A better way is to define a macro. This allows procrastination on deciding the format of paragraphs and facilitates change at some later date without a major editing effort.

Here is the paragraph macro used in this document:

```
.# pp --- begin paragraph
.de pp
.sp
.ne 2
.ti @[in]
.ti +5
.ns
.en pp
```

First a line is skipped via the 'space' command. Then, after checking that there is room on the current page for the first two lines of the new paragraph, a temporary indentation is set up that is five columns to the right of the running indentation with the two **ti** commands. Finally, no-space mode is turned on to suppress unwanted blank lines.

Sub-headings

Sub-headings such as the ones used here may be easily produced with the following macro:

```
.# sh --- sub-heading
.de sh
.sp 2
.ne 4
.ti @[in]
.bf
[1]
.pp
.en sh
```

First, two blank lines are put out. Then it is determined if there are four contiguous lines on the current page: one for the heading itself, one for the blank line after the heading, and two for the first two lines of the next paragraph. The temporary indentation value is then set to coincide with the current indentation value. Next, the first parameter passed to the macro (the text of the sub-heading) is boldfaced and a new paragraph is begun. The "pp" macro will insert the blank line after the heading.

Major Headings

Each section of this guide is introduced by a major heading that is boldfaced, underlined and centered on the page. The macro used to produce these headings is the following:

```
.# mh --- major heading
.de mh
.sp 3
.ne 5
.ce
.ul
.bf
[1]
.sp
.pp
.en mh
```

This is similar to the sub-heading macro: three blank lines are put out; a check for enough room is made; the parameter is centered, underlined and boldfaced; another blank line is put out; and a new paragraph is begun.

Tables of Contents

Table of contents can be automatically generated by writing the contents to a temporary file, then at the end reading that file to produce the table of contents. In the examples above we could divert subheadings and headings to a temporary file; e.g., add the following to the 'sh' and 'mh' macros. (These examples are similar to what is used to produce the table of contents of this guide; for pedagogical reasons we have simplified it a little).

```
.# generate a table of contents entry for a heading
.dv 5
.cc #
#sp
#ne 8
[bf [1]] @[tc]@[tcpn]
#cc .
.dv

.# table of contents entry for sub-heading
.dv 5
.cc #
#ne 4
   [1] @[tc]@[tcpn]
#br
#cc .
.dv
```

Each time a heading is printed a line is written to temporary file "5" containing the heading, boldfaced, followed by a blank, a tab and finally the current page number right justified in four columns. Each time a subheading is printed a line is written containing three blanks, the subheading, a blank, a tab and finally the current page number. Note that we precede diverted commands by a different control character because 'dv' will execute commands instead of diverting them.

The very last command of the document would be a generate table of contents macro, e.g.,

```
.# TC --- generate table of contents
.de TC
.cc #
#bp
#fo ..- @[rn @[pn]] -..
#ce "TABLE OF CONTENTS"
#rm -6
#ta @[rm]
#rm +6
#rc .
#ns
#so 5
#cc .
.en TC
```

This macro will set the control character to correspond to the control characters written to output stream "5," advance to the top of the next page, center the heading "TABLE OF CONTENTS", set the footer to print the page number in small roman numerals (the page number must be set prior to calling 'TC'), set the tab column to 6 columns to the left of the right margin (this generates 2 blanks followed by the page number which is right justified in four columns), sets the replacement tab character to "." and reads the contents of temporary file "5".

Quotations

Lengthy quotations are often set apart from other text by altering the left and right margins to narrow the width of the quoted text. Here is a pair of macros that may be used to delimit the beginning and end of a direct quotation:

```
.# bq --- begin direct quote
.de bq
.sp
.ne 2
.in +5
.rm -5
.lt +5
.en bq


.# eq --- end direct quote
.de eq
.sp
.in -5
.rm +5
.en eq
```

Notice the **lt** command in the first macro. To avoid affecting page headings and footings, the left margin is not adjusted; rather, an additional indentation is applied. But to increase the right margin width, there is no other alternative but to use the **rm** command. The 'title-length' command is thus necessary to allow headings and footings to remain unaffected by the interim right margin.

Italics

Since most printers can't easily produce italics, they are frequently simulated by underlining. The following macro 'italicizes' its parameter by underlining it.

```
.# it --- italicize (by underlining)
.de it
.ul
[1]
.en it
```

Boldfacing

While 'fmt' has built-in facilities for boldfacing, their use may be somewhat cumbersome if there are many short phrases or single words that need boldfacing; each phrase or word requires two input lines: one for the **bf** command and one for the actual text. The following macro cuts the overhead in half by allowing the command and the text to appear on the same line.

```
.# bo --- boldface parameter
.de bo
.bf
[1]
.en bo
```

Examples

This guide is peppered with examples, each one set apart from other text by surrounding blank lines and additional indentation. The next two macros, used like the "bq" and "eq" macros, facilitate the production of examples.

```
.# bx --- begin example text
.de bx
.sp
.ne 2
.nf
.in +10
.en bx
```

```
.# ex --- end example text
.de ex
.sp
.fi
.in -10
.en ex
```

Note that the definition of the "ex" macro causes the **ex** command to become inaccessible.

Table Construction

One example of table construction (for a table of contents) has already been mentioned in the section dealing with tabs. Another type of table that occurs frequently is that used in the command summaries in this guide. Each entry of such a table consists of a number of 'fields', followed on the right by a body of explanatory text that needs to be filled and adjusted.

The easiest way to construct a table like this involves using a combination of tabs and indentation, as the following series of commands illustrates:

```
.in +40
.ta 14 24 34 41
.tc \
```

The idea is to set a tab stop in each column that begins a field, and one last one in the column that is to be the left margin for the explanatory text. The extra indentation moves the effective left margin to this column. To begin a new entry, temporarily

undo the extra indentation with a **ti** command, and then type the text of the entry, separating the fields from one another with a tab character:

```
.ti -40
field 1\field 2\field 3\field 4\Explanatory text
```

The first line of the entry will start at the old left margin. Then all subsequent lines will be filled and adjusted between column forty-one and the right margin (inclusive).

Subsystem Macro Packages

Introduction

The previous section discussed how you might go about writing macros which do all kinds of nifty things, including building a table of contents. Fortunately, you do not have to write your own macro packages, since the Subsystem comes with several already written.

The two major packages are the User Guide Macros, and the Report macros. The Report macros are an older set of macros; their use is discouraged in favor of the User Guide Macros, which can actually be easily adapted for almost any kind of paper you may have to write. Users who wish to use the Report macros may print them off to see what they do and how they work. They are in =fmac=/report and =fmac=/ds_report for single- and double-spaced reports, respectively.

There are also macros for formatting Master's and Ph.D. theses. These are contained in =fmac=/gt_thesis. They are meant to be used by themselves, without any of the =fmac=/ev?* files (discussed below). The macros are documented in the file itself; see there for details on using them. You will probably want to change them to have your school's name, instead of Georgia Tech.

Accessing The User Guide Macros

To use the User Guide Macros in your paper, you may name them on the command line, or more conveniently, use one of the lines

```
.so =fmac=/ugh
```

- or -

```
.so =fmac=/ugnh
```

as the first line in your 'fmt' input file. The first command provides you with a report that uses plain headings (like the ones in this guide), while the second provides you with numbered

headings (useful for technical reports). In either case, the macros are used in an identical fashion. You should not need to change the text of your document in order to get either numbered or plain headings; you just need to switch macro packages.

Each of these files sets up the macros for headings, and then does a

.so =fmac=/ugm

to include the rest of the User Guide macros.

Using The User Guide Macros

The User Guide macros will automatically produce a title page and table of contents. The macros and their functions are:

.TP	Start the Title Page.
.AU	List the name(s) of the author(s).
.PD [<date>]	Give the publication date.
.CH [<heading text>]	Chapter heading.
.MH [<heading text>]	Major heading (within a chapter).
.SH [<heading text>]	Sub-heading (within a major heading).
.PH [<heading text>]	Paragraph heading (within a sub-heading).
.pp	Start a new paragraph (do not use after .PH).
.bq [<length>]	Begin an indented quote.
.eq	End an indented quote.
.be [<length>]	Begin an example.
.ee	End an example.
.ep	Skip to an even page.
.op	Skip to an odd page.
.HI	Produce a hanging Indent. Used for lists like this one.

.TC	Generate the table of contents (reset the page number with a **.bp** n, first).

So, a full paper might look something like this:

```
.TP
On The Preservation Of The Arithmetic IF
.AU
Arnold D. Robbins
Eugene H. Spafford
.PD "[ldate]"
.op
.HE "Saving The Arithmetic IF"
.#  The .HE macro will be explained shortly
.fo ''- # -''
.CH "Chapter 1"
...
.MH "Major 2"
...
.SH "Sub 3"
...
.PH "Par 4"
...
.bp 3
.TC
```

The title page produced would look just like the title page of this guide. You may want to change the **.PD** macro in =fmac=/ugm to have the name and address of your school or business, instead of Georgia Tech.

The heading macros each use two additional macros; one to help generate the table of contents, and one to actually produce the heading. For instance, **.CH** calls **.Ch** to produce the table of contents entry, and **.ch** to produce the chapter heading. The other header macros are implemented in a similar fashion. It is occasionally useful to access these macros directly; for instance in order to produce a foreword to a document, without having the foreword show up in the table of contents.

You should use all the .?H macros when writing your papers, i.e., the **.CH** macro, as well as the **.MH** and **.SH** macros. If you do not use the **.CH** macro, and you wish to use the numbered headings macros, your major sections will be sections 1, 2, 3, ... of Chapter 0, not Chapter 1, so bear this in mind.

It is never necessary to use a **.pp** macro after any of the heading macros, since they all do a **.pp** for you. In particular, the **.PH** heading macro should not be followed by a **.pp**; while after the other macros a **.pp** will only cause an extra line to be skipped.

The **.be** and **.bq** macros each take an optional argument, which is the the length of the example or quote. For a small quote or

example, you probably do not need to provide the length.

Since your entire document has to be formatted before the table of contents can be produced, the **.TC** macro should come at the end of your paper. You need to do a **.bp** n to the proper page for the table of contents (usually n = 3). The macros use diversion stream number five for the table of contents, so you should not use stream five for doing any of your own diversions.

The Printing Environment And The .HE Macro

The User Guide macros are designed so that a paper which uses them may be formatted on a variety of output devices, without changing the text of the paper. This is done by defining the printing environment in a macro; specifically the **.EV** macro. This macro takes care of setting the margin values, the page and margin offsets, the even and odd offsets, and the page length, among other things.

There are different environment files for different output devices. The files and the environments they are designed for are:

=fmac=/evd	Format output for the Diablo.
=fmac=/evp	Format output for the line printer.
=fmac=/evl	Format output for the Georgia Tech Xerox 9700 laser printer (See the help on 'lz'). These macros are for the User Guides.
=fmac=/evl2	Format output for the Georgia Tech Xerox 9700 laser printer. These macros are for the Reference Manual.
=fmac=/evt	Format output for "typesetting" on the Spinwriter. The output produced is designed to be photo-reduced to 8 1/2" by 11".

Unless you are positive that you will always use a particular output device, these files should not be included in your 'fmt' input file. Instead, they should be named on the command line. The **.TP** macro automatically calls the **.EV** macro to reset the environment.

The ev? files also define the **.HE** macro, which is used for designating the page headings. For single sided output, **.HE** is:

```
.de HE <left> <center> <right>
@[cc]he `[1]`[2]`[3]`
.en HE
```

while for double sided output (like the printed user guides), **.HE** is:

```
.de HE <left> <center> <right>
@[cc]eh `[1]`[2]`[3]`
@[cc]oh `[3]`[2]`[1]`
.en HE
```

The **.HE** macro should be placed right after the **.bp 1** command for the first page of your document, and before the first **.CH** command.

There is no special macro for footers. They are left to the **.fo** command. The usual choice is:

```
.fo ''- # -''
```

which places the page number at the bottom of the page.

There are environment files for the Report macros as well. The files are =fmac=/envd and =fmac=/envp for the Diablo and line printer, respectively.

Conclusion

The macros available to you with the Subsystem should satisfy most of your documentation needs, particularly with the variety of output devices that are supported. They can also be easily changed to suit your requirements, since the source files for the macro packages are included with the Subsystem.

Summary of Commands Sorted Alphabetically

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.#	-	-	no	Introduce a comment.
.ad c	both	both	no	Set margin adjustment mode.
.am xx	-	-	no	Add additional text to the body of a previously defined macro.
.bf N	N=0	N=1	no	Boldface N input text lines.
.bp ±N	N=1	next	yes	Begin a new page.
.br	-	-	yes	Force a break.
.c2 c	`	`	no	Set no-break control character.
.cc c	.	.	no	Set basic control character.
.ce N	N=0	N=1	yes	Center N input text lines.
.de xx	-	ignored	no	Begin definition or redefinition of a macro.
.dv <stream>	-	end '.dv'	no	Temporarily divert the output stream to a "filename" or to a temporary file designated by an integer "N" (to be later read by a ".so N" command) until a 'dv' command with no arguments is seen.
.ef /l/c/r/	blank	blank	no	Set even-numbered page footing.
.eh /l/c/r/	blank	blank	no	Set even-numbered page heading.
.en xx	-	ignored	no	End macro definition.
.eo ±N	N=0	N=0	yes	Set even page offset.
.er text	-	ignored	no	Write a message to the terminal.

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.ex	-	-	yes	Exit immediately to the Subsystem.
.fi	on	-	no	Turn on fill mode.
.fo /l/c/r/	blank	blank	no	Set running page footing.
.he /l/c/r/	blank	blank	no	Set running page heading.
.hy	on	-	no	Turn on automatic hyphenation.
.if <args>	-	ignored	maybe	Conditional execution of an input line.
.in ±N	N=0	N=0	yes	Indent left margin.
.lm ±N	N=1	N=1	yes	Set left margin.
.ls N	N=1	N=1	no	Set line spacing.
.lt ±N	N=60	N=60	no	Set length of header, footer and titles.
.m1 ±N	N=3	N=3	no	Set top margin before and including page heading.
.m2 ±N	N=2	N=2	no	Set top margin after page heading.
.m3 ±N	N=2	N=2	no	Set bottom margin before page footing.
.m4 ±N	N=3	N=3	no	Set bottom margin including and after page footing.
.mc <char>	BLANK	BLANK	no	Set margin character.
.mo ±N	N=0	N=0	no	Set margin offset.
.na	-	-	no	Turn off margin adjustment.
.ne N	-	N=1	yes	Express a need for N contiguous lines.
.nf	-	-	yes	Turn off fill mode. (Also inhibits adjustment.)
.nh	-	-	no	Turn off automatic hyphenation.

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.ns	on	-	no	Turn on 'no-space' mode.
.nx file	-	next arg	no	Move on to the next input file.
.of /l/c/r/	blank	blank	no	Set odd-numbered page footing.
.oh /l/c/r/	blank	blank	no	Set odd-numbered page heading.
.oo ±N	N=0	N=0	yes	Set odd page offset.
.pl ±N	N=66	N=66	no	Set page length.
.pn ±N	N=1	ignored	no	Set page number.
.po ±N	N=0	N=0	yes	Set page offset.
.ps N M	N=M=0	N=M=0	yes	Skip pages while (page number mod M) is less than N.
.rc c	BLANK	BLANK	no	Set tab replacement character.
.rm ±N	N=60	N=60	yes	Set right margin.
.rs	-	-	no	Turn off 'no-space' mode.
.sb	off	-	no	Single blank after end of sentence.
.so <stream>	-	ignored	no	Temporarily alter the input source. "Stream can be a "-" to indicate standard input, a "filename," or an integer "N" corresponding to a temporary file created by a previous '.dv N' command.
.sp N	-	N=1	yes	Put out N blank lines.
.ta N ...	9 17 ...	all	no	Set tab stops.
.tc c	TAB	TAB	no	Set tab character.
.ti ±N	N=0	N=0	yes	Temporarily indent left margin.

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.tl 'l'c'r'	blank	blank	yes	Generate a three part title.
.ul N	N=0	N=1	no	Underline N input text lines.
.xb	on	-	no	Extra blank after end of sentence.