TEXTFORM Reference Manual

January 1986

The University of Michigan
Computing Center
Ann Arbor, Michigan

Acknowledgement: This manual was written by the staff of Computing Services at the University of Alberta, Edmonton, Alberta, Canada. Minor changes have been made to reflect the differences between facilities at the two universities.

Table of Contents

Before You Begin . . . 1
- Why Use TEXTFORM? . . . 1
- A TEXTFORM Example . . . 1

Basic TEXTFORM . . . 5
- Starting Lines . . . 5
- Leaving Extra Space Between Lines . . . 7
- Paragraphs . . . 8
- Starting Pages . . . 8
- Footnotes . . . 8
- Lists of Points . . . 9
- Underlining and Italics . . . 10
- Spacing Words on the Line . . . 11

TEXTFORM Language in Detail . . . 25
- What is Computer Text Formatting? . . . 25
- Text Processing Concepts . . . 25
- How Does TEXTFORM Work? . . . 26
- Items That Can Appear in Command Mode . . . 27
- Characters Which Have Special Uses in Text Mode . . . 29
- Lengths in TEXTFORM . . . 31
- Notation Used For Subsequent Parts of This Manual . . . 32
- User-Defined Names in TEXTFORM . . . 34
- Choose an Output Device for the Document . . . 34
- Getting a Proof or Working Copy of a Document . . . 35
- Sample Proof . . . 36

Character Appearance . . . 37
- Underlining . . . 37
- Fonts . . . 38
- Typeface or Character Set . . . 39
- Producing Special Characters . . . 40
- Superscripts and Subscripts . . . 40
- Overstruck Characters . . . 41
- Overstriking Characters without Centring . . . 42
- Automatically Replacing Text Characters in the Input . . . 42
- Typesize . . . 43
- Other Modifications to Text Appearance . . . 44

Page Appearance . . . 46
- Size of Page . . . 46
- Margins . . . 46
- Page Position Commands . . . 47
- Blank Pages . . . 47
- Front and Back Pages . . . 48
- Page Numbers . . . 49
- Keeping Text on the Same Page or Column . . . 50
- Vertical Position of Text . . . 51
- Commands Which Include Input at Special Positions . . . 54

Producing Columns on the Page . . . 74
- Logical Pages . . . 75
- Tables . . . 82
- Defining Specific Columns in Logical Pages and Tables . . . 91
- Displaying the Logical Page or Table Dimensions . . . 92
- Producing Columns Without Using DEFINE TABLE . . . 93

Variables . . . 95

Predefined Variables 95
System Variables 95
User-Defined Variables 95
Values that can be Assigned to Variables 97
Operators 107
Expressions 112
Macros 113
Modifying the Action of the Macro When it is Used 113
Changing the Macro 116
Text Within Macros 116
Sample Form Letter Macro 116
More Control over Macros 117
Details about the Macro Definition and Use 119
Information about User-Defined Names 122
Erasing Names 122
Checking Whether Name Exists 122
Checking Type of a Name 122
Giving Attributes to a Name 123
Information About the Attributes of a Name 127
Table of Contents and Index 129
Table of Contents 129
Index 135
Line Drawing 144
Specifying Position of Line 144
Drawing lines 145
Program Control with TEXTFORM 149
System Variables 149
System Constants 149
Information About Current Page Position 150
Formatting Information 151
Including Input If a Condition is True or False 156
Including Input from a List of Choices 158
Including Input WHILE a Condition is True 159
Including Input FOR a Specific Number of Times 159
Functions 160
Calling TEXTFORM as a Subroutine 163
TEXTFORM and MTS 167
The RUN Command in Detail 167
Ending the TEXTFORM Run 168
Errors Which End the TEXTFORM Run 169
System Information During the Run 169
Output on SERCOM 170
Output on SPRINT 172
Running TEXTFORM Interactively by Giving GUSER Input 177
Interrupts in the TEXTFORM Run 179
Appendix 1 – TEXTFORM Language 184
Appendix 2 – TEXTFORM's Internal Storage 236
Index 237
Index to TEXTFORM Language 246
Error Messages 251

BEFORE YOU BEGIN

Why Use TEXTFORM?

TEXTFORM can produce a variety of documents, ranging from single-page memos and letters to entire books. It is primarily aimed at the production of multi-page documents, since it may be too costly for very short documents.

A computer text formatting system has a major advantage over conventional document production. Once the contents of the document have been entered into the computer, they can easily be modified. The powerful MTS File Editor allows you to revise the contents of a document without having to be concerned about the format. With the TEXTFORM program, you can change the format without affecting the contents. Errors that are inevitably introduced when a document is retyped are thus eliminated, and revisions and updates are no longer onerous tasks.

It is important that you understand that there generally is not a one-to-one correspondence between the lines you type and the lines of your document. One **input line** (a line you have typed into your file) may generate several **output lines** (lines on the final page), or several input lines may be needed to generate a single output line.

A TEXTFORM Example

Create the Input File

MTS and File Editor commands must be used to create a file and insert into it the text that is to be formatted. The following example of a session at a terminal illustrates this. In this example, the characters #, ?, and : are produced by MTS; do not type them. Italics indicate information specific to you.

```
# signon abcd
? password
# create trial
File "TRIAL" has been created.
# edit trial
: insert
?The earliest computer programs were written in a
?rudimentary system of notation called machine language.
?The numerical codes of machine language were soon
?replaced by the mnemonic codes of a slightly higher-level
?language called assembly language.
?A separate program called an assembler was employed to
?transcribe assembly-language instructions into the machine
```

```
?codes that could be executed directly by the computer.
?Machine-language and assembly-language programs are
?detailed and repetitious.
?
: stop
#
```

The complete file looks like this:

```
# list trial
>    1  The earliest computer programs were written in a
>    2  rudimentary system of notation called machine language.
>    3  The numerical codes of machine language were soon
>    4  replaced by the mnemonic codes of a slightly higher-level
>    5  language called assembly language.
>    6  A separate program called an assembler was employed to
>    7  transcribe assembly-language instructions into the machine
>    8  codes that could be executed directly by the computer.
>    9  Machine-language and assembly-language programs are
>   10  detailed and repetitious.
# End of file
```

Here are a few tips to keep in mind when entering the input text:

- Since TEXTFORM normally recognizes only the first blank between words, make generous use of blanks to make the input file as readable as possible. Indent portions of the input text that have special meaning and separate sections by blank lines provides for greater readability when editing files. This helps locate certain portions of the input text for later revision.
- Keep the lines short—type 60 characters or less per line. TEXTFORM produces an error if an input line exceeds 256 characters.
- Start each new sentence on a new line, to make editing easier.
- Be consistent in the spelling, capitalization, and placement of TEXTFORM instructions within the input text, again, for ease of editing.

Run the TEXTFORM Program

To have the TEXTFORM program format the text, give an MTS command of the following form:

```
# run *textform scards=file spunch=result
```

From this model, replace *file* with the name of the file which contains the text, and replace *result* with the name of an empty file, which can be a temporary file, as in:

```
# run *textform scards=file spunch=-res
```

Print the Formatted Text

If for some reason there are TEXTFORM errors in your file, error messages may appear at your terminal after you type the run command. If this happens, read the next section on error messages before printing your document.

After the run command, the formatted text is in the file *–res*. To print it on the Xerox 9700:

```
# run *pagepr scards=-res
# Execution begins
  *PRINT* assigned receipt number 622643
  *PRINT* 622643 held
  This *PAGEPR run generated 406 lines, 5 pages, 5 images,
  and 3 sheets.
  *PRINT* 622643 released, 7 pages, route=CNTR,
  printer=PAGE.
# Execution terminated
```

The formatted text will appear on an 8.5 inches by 11 inches page, single-spaced, with four 1-inch margins (although the following example is somewhat narrower, and in a different typeface):

> The earliest computer programs were written in a rudimentary system of notation called machine language. The numerical codes of machine language were soon replaced by the mnemonic codes of a slightly higher-level language called assembly language. A separate program called an assembler was employed to transcribe assembly-language instructions into the machine codes that could be executed directly by the computer. Machine-language and assembly-language programs are detailed and repetitious.

How to Interpret Error Messages

The following sample file contains several TEXTFORM instructions, or commands. The commands can be entered in upper or lower case.

```
# list trial
>   1  The earliest computer programs were written in a
>   2  rudimentary system of notation called machine language.
>   3  <LINE>1. The numerical codes of machine language were soon
>   4  replaced by the mnemonic codes of a slightly higher-level
>   5  language called assembly language.
>   6  <LINE>2. A separate program called an assembler was used
>   7  to transcribe assembly-language instructions into the
>   8  machine codes that could be executed directly by the
>   9  computer. Machine-language and assembly-language programs
>  10  are detailed and repetitious.
# End of file
```

If you enter a command that TEXTFORM does not recognize, perhaps as the result of a typing error, TEXTFORM responds with an error message which it prints at your terminal. For example, if line 6 of the file contained:

<LNE>2. A separate program called an assembler was used to

the following messages would appear at your terminal after you typed the RUN command:

```
1   4   6   <LNE>2. A separate program called an assembler was used
to
Error        25: LNE is not defined. Ignored.
in column        1     "LNE>2. A separate"
```

The information in the message helps you find the error in your file. The numbers 1, 4 and 6 on the first line indicate that the error occurred on page 1 of the document, line 4 on the page, and that the input line containing the error was line 6 of the file. All error messages are numbered; this is message 25. The 'in column 1' tells you that TEXTFORM found the errant command starting at column 1 of input line 6. This lets you go back into your file, correct the error, and rerun TEXTFORM.

In this manual, messages and information displayed by TEXTFORM appear in italics. If you want to print a copy of these messages, set SERCOM in the RUN command:

run *textform scards=*file* spunch=–*res* sercom=–*error*

Now the messages are stored in the file –*error*, which you can edit or copy to *print*.

BASIC TEXTFORM

This section of the manual describes basic TEXTFORM instructions. Once you have read it and tried the examples, you will be able to produce a simple document. The rest of the manual describes the TEXTFORM command language in more detail.

Previous examples showed that an input file containing only text can be processed by TEXTFORM. This is because TEXTFORM has **defaults** —actions which take place unless you specify otherwise. To get a more specific format, a document must have TEXTFORM instructions inserted into the input file, along with the text.

Starting Lines

The most frequently used TEXTFORM commands are LINE and LINEEND, abbreviated as L and LEND. They are used when a new line must be started in the document, since a new line in the input file does not start a new output line in the document.

```
The earliest computer programs were written in a
rudimentary system of notation called machine language.
<LINE>1. The numerical codes of machine language were soon
replaced by the mnemonic codes of a slightly higher-level
language called assembly language.
<LINE>2. A separate program called an assembler was used to
transcribe assembly-language instructions.
```

produces:

> The earliest computer programs were written in a rudimentary system of notation called machine language.
> 1. The numerical codes of machine language were soon replaced by the mnemonic codes of a slightly higher-level language called assembly language.
> 2. A separate program called an assembler was used to transcribe assembly-language instructions.

Each time the LINE command is used, TEXTFORM ends the current line, and spaces down to start the next line. LINE makes sure that TEXTFORM is at the start of a line, and has no result if this is already the case. Because of this, LINE, LINE is the same as LINE.

The vertical spacing between lines is determined by LINESPACE. By default, it is 1/6 of an inch, about .16 inches. On most devices, this gives the impression of 'single spacing'. If you require 3 lines per inch, change LINESPACE:

```
<LINESPACE = .3INCH>
```

Changes to LINESPACE depend on the capabilities of the output device. For example, if you want 5 lines per inch, and set LINESPACE to .2 inches, the device may be unable to produce this. It will then print 6 lines per inch, although LINESPACE is still .2 inches.

Each time a line begins, TEXTFORM spaces down by the value in LINESPACE. Therefore, when a LINE command is given, TEXTFORM moves LINESPACE down from the previous line of text. A LINESPACE change after a LINE command then affects subsequent lines.

```
<LINESPACE = .33INCH>
The earliest computer programs were written in a
rudimentary system of notation called machine language.
<LINE, LINESPACE = .16INCH>
1. The numerical codes of machine language were soon
replaced by the mnemonic codes of a slightly higher-level
language called assembly language.
<LINE>2. A separate program called an assembler was used to
transcribe assembly-language instructions.
```

produces:

> The earliest computer programs were written in a rudimentary system
>
> of notation called machine language.
>
> 1. The numerical codes of machine language were soon replaced by the mnemonic codes of a slightly higher-level language called assembly language.
> 2. A separate program called an assembler was used to transcribe assembly-language instructions.

If you want the *next* line to use the new value of LINESPACE, change LINESPACE *before* the next line:

```
1. The numerical codes of machine language were soon
replaced by the mnemonic codes of a slightly higher-level
language called assembly language.
<LINE>2. A separate program called an assembler was used to
transcribe assembly-language instructions.
<LINEEND, LINESPACE = .33INCH>
The earliest computer programs were written in a
rudimentary system of notation called machine language.
```

produces:

> 1. The numerical codes of machine language were soon replaced by the mnemonic codes of a slightly higher-level language called assembly language.
> 2. A separate program called an assembler was used to transcribe assembly-language instructions.
>
> The earliest computer programs were written in a rudimentary system

of notation called machine language.

The second line-positioning command, called LINEEND, can be used to end the current line without spacing down to the start of the next line. Instead of LINESPACE=.33INCH, LINE you can say LINEEND, LINESPACE=.33INCH.

LINEEND ends a line only if one has been started. Since the command LINE 'starts' a line, the commands LINE, LINEEND produce an empty line. If you want extra vertical space between lines or at the top of a page, use the VERTSPACE command, described next.

Leaving Extra Space Between Lines

The vertical space generated by the LINE command is always LINESPACE high. The VERTSPACE, or VS command, spaces down immediately by the length specified. It does not change the horizontal position. The following example leaves half an inch of white space, and starts a line:

Half<VERTSPACE .5INCH, LINE>an inch

produces:

Half

an inch

As with changes to LINESPACE, the amount of vertical space that appears may not be exactly what you requested if the output device cannot produce it. The vertical position of text following this command may be adjusted upwards by specifying a negative length. In this manner, superscript and subscript characters can be produced, as described on page 40.

Up <VERTSPACE –.16INCH>a<VERTSPACE .16INCH>bit

produces:

a
Up bit

More details about vertical spacing on the page are given on page 51.

Paragraphs

The command to begin a new paragraph is NEWPARA, or NP. This command ends the current line, begins a new line, and spaces the first line in the paragraph in .5 inch from the left margin:

> Ask not for whom the bell tolls; it tolls for thee.
> <NEWPARA> Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.

produces:

> Ask not for whom the bell tolls; it tolls for thee.
>
> Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.

TEXTFORM will not begin a new paragraph on the last line of the page. If there is only one line left when NEWPARA is encountered, a new page is started before the paragraph is formatted.

By default, paragraphs are indented by .5 inch. If you want to change this, change PARAIND, usually at the beginning of your source file:

> <PARAIND = 1INCH>

If you do not want the blank line that separates paragraphs from the preceding text, enter:

> <PARASEP = 0INCH>

Starting Pages

When the current page is full of text, TEXTFORM automatically starts the next page. However, the command PAGE, or P, forces the immediate start of a page, even when the previous page is not full of text. More details about the appearance of the page are given starting on page 46.

Footnotes

The FOOTNOTE, or FOOT, command lets you enter a footnote without knowing where you are in relation to the bottom of the page. Text is saved and printed at the bottom of the page, and is numbered with superscript arabic numerals. Enter the footnote immediately after you refer to it in the text, without a space before the FOOTNOTE command. End the text of the footnote with ENDFOOTNOTE, or EFOOT:

> reference in the text.<FOOTNOTE>This footnote is at the bottom of the page. <ENDFOOTNOTE> Text continues

produces in the text:

> reference in the text.† Text continues

The footnote appears at the bottom of this page. Notice footnotes are separated from the rest of the text on the page with a separating line. Complete details about footnotes are given on page 56.

Lists of Points

TEXTFORM provides an automatic facility for numbering lists. The command to do the numbering is PT. The list is ended with EPT.

```
which fall into the following groups:
<PT> Slightly modified classical histological techniques with fluid
fixation, wax embedding, and aqueous mounting.
<PT> Sandwich technique with separate processing of tissue and
photographic film after exposure.
<PT> Protective coating of tissue to prevent leaching during
application of stripping film or liquid emulsion.
<EPT> One can also mount the frozen sections on emulsion, using
heat or adhesive liquids.
```

produces:

> which fall into the following groups:
> 1. Slightly modified classical histological techniques with fluid fixation, wax embedding, and aqueous mounting.
> 2. Sandwich technique with separate processing of tissue and photographic film after exposure.
> 3. Protective coating of tissue to prevent leaching during application of stripping film or liquid emulsion.
>
> One can also mount the frozen sections on emulsion, using heat or adhesive liquids.

The format of the list can be modified in many ways, as described in the PT description on page 219. One of the changes produces the list without numbering. This format is often used for bibliographies.

```
which fall into the following groups:
<PT(HANG)> Slightly modified classical histological techniques with
fluid fixation, wax embedding, and aqueous mounting.
<PT> Sandwich technique with separate processing of tissue and
photographic film after exposure.
<PT> Protective coating of tissue to prevent leaching during
application of stripping film or liquid emulsion.
<EPT> One can also mount the frozen sections on emulsion, using
heat or adhesive liquids.
```

produces:

----

†This footnote is at the bottom of the page.

which fall into the following groups:
Slightly modified classical histological techniques with fluid fixation, wax embedding, and aqueous mounting.
Sandwich technique with separate processing of tissue and photographic film after exposure.
Protective coating of tissue to prevent leaching during application of stripping film or liquid emulsion.
One can also mount the frozen sections on emulsion, using heat or adhesive liquids.

If you always want extra spacing before, between or after lists of points, this can be done automatically.

which fall into the following groups:
<PTPREGAP = PTPREGAP + .16INCH>
<PTPOSTGAP = PTPOSTGAP + .16INCH>
<PT> Slightly modified classical histological techniques with fluid fixation, wax embedding, and aqueous mounting.
<PT> Sandwich technique with separate processing of tissue and photographic film after exposure.
<PT> Protective coating of tissue to prevent leaching during application of stripping film or liquid emulsion.
<EPT> One can also mount the frozen sections on emulsion, using heat or adhesive liquids.

produces:

which fall into the following groups:

1. Slightly modified classical histological techniques with fluid fixation, wax embedding, and aqueous mounting.
2. Sandwich technique with separate processing of tissue and photographic film after exposure.
3. Protective coating of tissue to prevent leaching during application of stripping film or liquid emulsion.

One can also mount the frozen sections on emulsion, using heat or adhesive liquids.

Underlining and Italics

Text can be emphasized with underlining and italics. The command UNDERLINE controls underlining, and the command FONT is used for italics, as in the following examples. More details on these commands, and other ways to change character appearance, are given on page 37.

Titles of publications, such as <FONT 2>TEXTFORM Manual<FONT> are usually italicized, although some authors use underlining, as in <UNDERLINE ON>TEXTFORM Manual<UNDERLINE OFF>.

produces:

Titles of publications, such as *TEXTFORM Manual* are usually italicized, although some authors use underlining, as in TEXTFORM Manual.

Spacing Words on the Line

TEXTFORM recognizes a **word** as any non-blank character or characters followed by one or more blanks, or followed by the end of the input line. (A **character** is a letter, numeral, symbol, or mark of punctuation.) The following line contains 5 words:

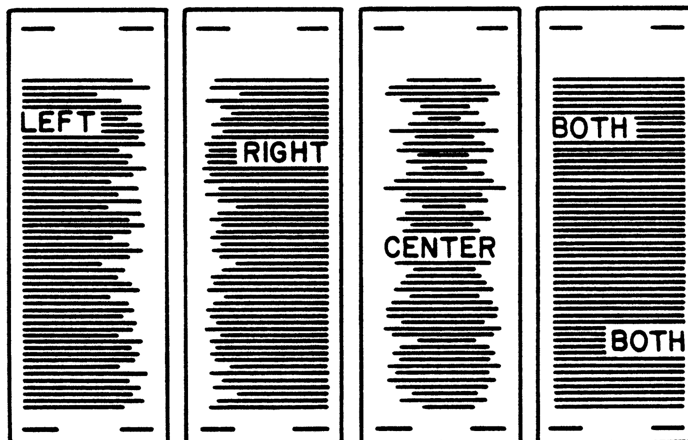
word 1889 1982-1982   the   end

The word 'end' does not need a blank after it because it is at the end of the input line.

TEXTFORM places the text, word by word, on the formatted page with one space between each word (this is the word space). When a word does not fit at the end of the line TEXTFORM begins a new line. If there are too many characters in one word to fit within the margins, TEXTFORM produces the error *Unable to keep non-line-breaking words within the margins* and then prints the word. This problem can be corrected by indicating where TEXTFORM may hyphenate or break the word, as described on page 22.

Alignment of Text

Alignment is a term describing the way text is placed in relation to the current margins. When text is aligned to the left margin, it is said to be left-aligned, and lines are different lengths. This is the default. When text is aligned to both margins, or **justified**, extra spaces are inserted between the words. If you want to change the way text is aligned, change ALIGNMENT to one of:



For example, to centre text, use:

<ALIGNMENT = CENTER>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom

> the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.
> <LINEEND, ALIGNMENT=LEFT>

produces:

> Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.

If you are changing ALIGNMENT for only several lines, end the line of text *before* resetting ALIGNMENT back to its previous value (because TEXTFORM only checks the value of ALIGNMENT at the end of each line). In the above example, it was ended with a LINEEND command. Any other command which ends the line, such as NEWPARA, could have been used.

If ALIGNMENT=BOTH, lines are justified only when text overflows onto the next line. TEXTFORM does not justify the line which is ended as the result of a command such as LINE, LINEEND, or NEWPARA.

Although justified text may seem easier to read, this has not proven to be the case. Extra spaces must be inserted between words to make the lines the same length; this may be distracting to the reader, especially when line lengths are short.

If you want to be warned whenever a large justification space has been inserted, you can reduce the value of MAXWORDSPACE. When justifying, TEXTFORM will not insert a space greater than MAXWORDSPACE, which is initially 5 inches. To be warned of a justification space greater than .5 inches, enter in your file:

> <MAXWORDSPACE = .5INCH>

When a larger space is inserted, TEXTFORM produces the warning *Justification space required is greater than MAXWORDSPACE. MAXWORDSPACE used.* You can then make adjustments, perhaps by indicating where TEXTFORM can hyphenate or break the word, as described on page 22. TEXTFORM does not do automatic letter spacing.

Changing the Alignment of a Single Line

In most cases the same alignment is used throughout a document, although special lines may require a different alignment than that of the main body of text. Alignment can be changed, on a one-time basis, in the LINEEND command, by specifying the type of alignment required. This is simpler than resetting ALIGNMENT.

> <LINE> Yours truly, <LINEEND RIGHT>

produces:

Yours truly,

This one-time change to the LINEEND command affects the output line only. This

may be misleading, if one input line produces two output lines. The first output line, which overflows normally, uses ALIGNMENT. Only the second line uses the alignment given on the LINEEND command. When specifying alignment on the LINEEND command, it can be given as:

CENTRE or C
JUSTIFY or BOTH or J or B
LEFT or L
RIGHT or R

In this command, the alignment is a **keyword** to the LINEEND command. A keyword further describes how a command acts. When a command has one or more keywords, as this one does, the keywords are separated by blank spaces, and not commas. The comma is used between distinct commands, not parts of the same command.

Changing the Line Length with Indents

The size of a page and its margins cannot be changed once the page is begun. However, an **indent**, which temporarily changes the margins, can be used. The command to do this is INDENT, or I.

This discussion deals initially with indents from the left margin. The sizes of the left indents are in a list called LEFTINDENTS. By default, the list contains:

(.4IN, .8IN, 1.2IN, . . . , 8IN)

To use the first left indent, which is .4 inches, use:

<INDENT LEFT 1>

This INDENT LEFT command positions TEXTFORM .4 inches (the first value in the LEFTINDENTS list) in from the left margin. When INDENT LEFT 1 is in effect, each line of text appears not closer than .4 inches to the left margin. To turn the indent off, any of the following commands can be used:

<INDENT LEFT OFF> or
<INDENT LEFT 0> (zero) or
<INDENT LEFT>

These commands set the indent to zero; subsequent text begins on a new line at the left margin.

If the current text position is to the left of the indent requested in an INDENT command, the current text position stays on the same line. If the current text position is to the right of the requested indent, TEXTFORM must start a new line to get to that point. Since the actions differ depending on where the current text position is, most often, INDENT commands are given after a LINEEND command, as in the following examples:

```
<ALIGNMENT=BOTH>
<INDENT LEFT 4>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.
<LINEEND>
<INDENT LEFT 2>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.
<LINEEND>
<INDENT LEFT OFF>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.
```

produces:

> Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.

> Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.

Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee.

The right indent works similarly, using lengths in the list called RIGHTINDENTS.

```
<ALIGNMENT=BOTH>
<INDENT RIGHT 4>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.
<LINEEND>
<INDENT RIGHT 2>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.
<LINEEND>
<INDENT RIGHT OFF>
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.
```

produces:

Ask not for whom the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls for thee. Ask not

for whom the bell tolls; it tolls for thee. Ask not for
whom the bell tolls; it tolls for thee.
Ask not for whom the bell tolls; it tolls for thee. Ask not for
whom the bell tolls; it tolls for thee. Ask not for whom the
bell tolls; it tolls for thee. Ask not for whom the bell tolls; it
tolls for thee.
Ask not for whom the bell tolls; it tolls for thee. Ask not for whom
the bell tolls; it tolls for thee. Ask not for whom the bell tolls; it tolls
for thee. Ask not for whom the bell tolls; it tolls for thee.

Left and right indents can both be set together. The command:

<INDENT BOTH 2 3>

uses the second left and third right indents. If only one value is specified, such as INDENT BOTH 2, the second value is used from both LEFTINDENTS and RIGHTINDENTS. To turn off both indents in the same command, use:

<INDENT BOTH OFF>

The following lines indent a quotation:

The earliest computer programs were written in a rudimentary system
of notation called machine language. The numerical codes of machine
language were soon replaced by the mnemonic codes of a slightly
higher-level language called assembly language.
<LINE, INDENT BOTH 1>
A separate program called an assembler was used to transcribe
assembly-language instructions into the machine codes that could be
executed directly by the computer.
<LINEEND, INDENT OFF>
Machine-language and assembly-language programs are detailed and
repetitious.

produce:

The earliest computer programs were written in a rudimentary system
of notation called machine language. The numerical codes of machine
language were soon replaced by the mnemonic codes of a slightly
higher-level language called assembly language.
    A separate program called an assembler was used to
    transcribe assembly-language instructions into the machine
    codes that could be executed directly by the computer.
Machine-language and assembly-language programs are detailed and
repetitious.

If you want a single spaced quotation in double spaced text:

The earliest computer programs were written in a rudimentary system
of notation called machine language. The numerical codes of machine
language were soon replaced by the mnemonic codes of a slightly
higher-level language called assembly language.
<LINE, LINESPACE = .16IN, INDENT BOTH 1>
A separate program called an assembler was used to transcribe

assembly-language instructions into the machine codes that could be executed directly by the computer.
<LINEEND, INDENT OFF, LINESPACE = .33IN>
Machine-language and assembly-language programs are detailed and repetitious.

and produce:

The earliest computer programs were written in a rudimentary system

of notation called machine language. The numerical codes of machine

language were soon replaced by the mnemonic codes of a slightly

higher-level language called assembly language.

A separate program called an assembler was used to
transcribe assembly-language instructions into the machine
codes that could be executed directly by the computer.

Machine-language and assembly-language programs are detailed and

repetitious.

Changing the size of the indent

The sizes of the left and right indents are in two lists called LEFTINDENTS and RIGHTINDENTS. By default, these lists both contain:

(.4IN, .8IN, 1.2IN, . . . , 4IN)

If you require left indents of 1 inch and 2 inches, change LEFTINDENTS:†

<LEFTINDENTS = (1IN, 2IN)>

Although the lengths in the list don't need to be in ascending order, it is a good practice because they are easier to remember. When LEFTINDENTS or RIGHTINDENTS are changed, the actual indent in effect does not change until the next INDENT command. There are several other ways to change a list such as LEFTINDENTS. See examples on page 103.

Delayed Indents

The INDENT command takes effect immediately. However, if:

HANG=number

appears in the command, the specified indent is delayed until 'number' of lines appear in the output. This is called a **hanging** indent, and is often used in bibliographies. The following example produces the same results as PT(HANG),

----
†Changing LEFTINDENTS will also affect the indentation of lists formatted with the PT macro.

described earlier:

```
<INDENT LEFT OFF, LINESPACE = .16IN>
<INDENT HANG=1 LEFT 2>
Adams, Robert. “Langland and the Liturgy Revisited.” Studies in
Phililogy 73(1976):266-284.
<LINEEND, INDENT LEFT OFF, VERTSPACE .16IN>
<INDENT HANG=1 LEFT 2>
Alford, John Alexander. “A Note on Piers Plowman B.xviii. 390: ‘Til
Parce it Hote’.” Modern Philology 69(1972):323-325.
<LINEEND, INDENT LEFT OFF, VERTSPACE .16IN>
<INDENT HANG=1 LEFT 2>
Alford, John Alexander. “Some Unidentified Quotations in Piers
Plowman.” Modern Philology 72(1975):390-399.
<LINEEND, INDENT LEFT OFF, VERTSPACE .16IN>
```

produces:

Adams, Robert. “Langland and the Liturgy Revisited.” Studies in
Phililogy 73(1976):266-284.

Alford, John Alexander. “A Note on Piers Plowman B.xviii. 390: ‘Til
Parce it Hote’.” Modern Philology 69(1972):323-325.

Alford, John Alexander. “Some Unidentified Quotations in Piers
Plowman.” Modern Philology 72(1975):390-399.

Examples of Indents and Alignment

This example uses two values for ALIGNMENT on the same line. The item counters are centered and the items themselves are left aligned. Whenever an INDENT LEFT command is used to move to a new position on the line, TEXTFORM aligns the previous part of the line using the current value of ALIGNMENT:

```
<LINE, I L 0, ALIGNMENT=CENTRE>9.
<I L 2, ALIGNMENT=LEFT> This is the ninth item in a list where
the counters are centred.
<LINE, I L 0, ALIGNMENT=CENTRE>110.
<I L 2, ALIGNMENT=LEFT> This is an item later in the list.
```

produces:

9. This is the ninth item in a list where the counters are centred.
110. This is an item later in the list.

The next example modifies left and right indents on the same line. It shows that the INDENT LEFT command can be used as a temporary right margin. If ALIGNMENT=RIGHT, the commands I R 0, I L 6 treat LEFTINDENTS(6) as the right margin, and align text to the right to LEFTINDENTS(6). Then the command I L 8 causes LEFTINDENTS(8) to be the new temporary left margin.

```
<lend,i b 0>
<rightindents(10)=remaining(1)-leftindents(6)>
```

```
<lend,i b 0,i r 10,alignment=right>
In this brief example the
word that stands out is
<i r 0 , i l 6, i l 8, alignment=left, i h=1 l 10>
immediately apparent and is followed by more text here which flows
over several lines.
```

produces:

```
          In this brief
      example the word
      that stands out is    immediately apparent and is followed by more
                                    text here which flows over several
                                    lines.
```

Information about Indents

LINDENT and RINDENT contain the *lengths* of the current indents. For example, set up a left indent .5 inch greater than the current indent with:

```
< LEFTINDENTS(9) = LINDENT+.5IN >
```

LINDENTINDEX and RINDENTINDEX contain the current indent numbers. For example, after an INDENT RIGHT 3 command RINDENTINDEX contains 3. To indent to the next left indent, without knowing the current indent, give the command:

```
<INDENT LEFT LINDENTINDEX+1>
```

Any (perhaps unknown) pending hanging indents can be turned off using the following command, which re-issues the current indent settings:

```
<INDENT BOTH LINDENTINDEX RINDENTINDEX>
```

Horizontal Space within a Line

Each word is separated by the value of WORDSPACE. When you need a space wider than WORDSPACE, there are several options, described here.

By default, WORDSPACE is .1 inch, which is usually the width of a character. If you are using proportional characters, (where the characters have varying widths—the letter ‘m’ is much wider than the letter ‘l’) you may want to make WORDSPACE the same width as an average character. A good way to do this is to use TEXTWIDTH, which calculates the width of one or more characters.

```
<WORDSPACE = TEXTWIDTH('N')>
```

When text is being justified, TEXTFORM automatically adds extra space, in addition to word space, to make all lines the same length. You should make WORDSPACE somewhat smaller when proportional text is being justified, to compensate for the extra space that will be added.

```
<WORDSPACE = TEXTWIDTH('i')>
```

To control the maximum justification space that can be inserted, change MAXWORDSPACE as shown on page 12.

TEXTFORM also has a method of recognizing sentence endings, if you want extra spacing there. SENTSEP is the amount of space inserted at each sentence ending. In most cases, SENTSEP is set to be the same as WORDSPACE with the command:

<SENTSEP = WORDSPACE>

Extra Horizontal Space

The HORSPACE, or HS command inserts a horizontal space in the output which replaces the wordspace.

a<HORSPACE 1IN>b

produces:

a b

If blanks occur on either side of the HORSPACE command, they are ignored.

text1<HORSPACE 1IN>text2
text1 <HORSPACE 1IN> text2

both produce:

text1 text2

The horizontal space is underlined if UNDERLINEWORDSPACE is true (see underlining on page 37). Since HORSPACE replaces WORDSPACE, TEXTFORM may insert justification spaces where the HORSPACE command appears, in addition to the length requested, when ALIGNMENT=BOTH. If you do not want this, use BLANKCHARACTER, described below.

The length specified in a HORSPACE command may be positive or negative:

– If the horizontal space requested is greater than that remaining on the line, TEXTFORM goes to the next line (without aligning) and then produces the horizontal space.
– If the space requested is greater than the line width, an error message is given: *Requested horizontal space exceeds line width. One blank line produced.*
– If the length is negative, as in –1IN, the text following the HORSPACE command is positioned over the text which precedes the command. You cannot horizontally space into the margins.

  <LINE>abc<HORSPACE -1IN>

  produces the message *Negative horizontal space stopped at the left indent.*
– If the length is 0, no word space appears unless one is inserted by justification.

Horizontal Space for Characters

The command BLANKCHARACTER, or BC, leaves a blank space in the output which is treated as a character in a word.

```
A + B = C<BC>xy <BC> ***
```

produces:

```
A + B = C  xy     ***
```

Spaces which appear around the BLANKCHARACTER are treated as normal wordspaces. When underlining is on, the space produced by BC is underlined.

The space produced by BLANKCHARACTER is the width of the current character size (on devices that cannot change typesize this is usually .1 inch). A length can be given with the command to produce a larger space:

```
text<BLANKCHARACTER 1INCH>text
```

produces:

```
text                    text
```

The length may be negative, for example BC –1INCH.

When the characters are not all the same width, BC,BC,BC may not produce the same amount of space as three characters would:

```
abc This must line up
<LINE,BC,BC,BC> This must line up
```

produces:

```
abc This must line up
        This must line up
```

To produce a blank space the exact width of certain characters, use TEXTWIDTH, which calculates the width of the characters given, and inserts that width into the BC command:

```
abc This must line up
<LINE, BC TEXTWIDTH('abc')> This will line up
```

produces:

```
abc This must line up
    This will line up
```

Spacing Text Across a Line

The SPLIT, or SP command can be used to space text across a line. The commands:

<LINE> text <SPLIT> text <LINEEND>

produce:

text text

It is also a good idea to use commands to start and end the line to be split, to ensure exactly what is being split apart.

<LINE>abc <SPLIT> def <SPLIT> ghi <LINEEND>

produces:

abc def ghi

The SPLIT uses the characters in SPLITSTRING to split the parts of the line. By default, SPLITSTRING contains blanks, as in the previous example. To split a line with the character '.', change the value of SPLITSTRING and give the SPLIT command. Notice that blanks around the command are treated as wordspaces.

<SPLITSTRING = '.'>
<LINE> longer text <SPLIT> text <LINEEND>

produces:

longer text .... text

or supply the characters in the command itself, if you do not want those in SPLITSTRING used:

text<SPLIT '*/' >text

produces:

text /*/ text

The contents of SPLITSTRING, or the characters given in the SPLIT command are not interpreted by TEXTFORM, so there can be *no* commands in the split characters given.

TEXT<split 'a<f 2>b<f>' >TEXT

produces:

TEXT 2>b<f>a<f 2>b<f>a<f 2>b<f>a<f 2>b<f>a<f 2>b<f>a<f TEXT

The names of special characters can be used in the SPLIT commmand. For example, if the character EMDASH is available:

<LINE, SPLIT EMDASH, LINEEND>

produces:

The first complete split string is positioned an even multiple of the width of the string from the left margin, or current left indent. A portion of the split string, or horizontal space, may fill the space before the first complete split string. If the left indent changes, SPLIT works between the old and the new indent settings. The next example aligns numbers on the decimal point by changing left indents.

```
<LINE,I L 0,SP> 8.<I L 2,I L 3>First item in list is
<LEND> several lines long.
<LINE,I L 0,SP> 9.<I L 2,I L 3>Second item in list<SP '.'>
<LINE,I L 0,SP>10.<I L 2,I L 3>Third item in list.
```

produces:

```
 8.  First item in list is
     several lines long.
 9.  Second item in list ........................................................
10.  Third item in list.
```

Hyphenating and Breaking Words

TEXTFORM does not hyphenate or break any words, even those containing hyphens, unless it is instructed to do so. This can be done in several ways—by indicating where a word can be hyphenated with discretionary hyphens, by indicating where a word can be broken, or by turning on automatic hyphenation.

Discretionary Hyphen

A **discretionary hyphen** in a word is <–>, and indicates that the word may be hyphenated at that point, if necessary. You can use discretionary hyphens at any point within a document, even if automatic hyphenation is not in effect.

```
hy<–>phen<–>a<–>tion
```

produces, if the word overflows the line:

```
                                                            hyphen–
ation
```

Allow Line Break Without Hyphen

You may want to permit a long word to be broken only if it appears at the end of a line. The command to do this is ALLOWLINEBREAK, or ALB. In the following example, TEXTFORM could end the line with ‘major/’ and put ‘minor’ on the next line, if necessary. No hyphens are inserted:

```
major/<ALB>minor
```

produces:

major/minor

or, if the word overflows the line:

major/

minor

Automatic Hyphenation

TEXTFORM can do automatic hyphenation in one of two ways:

1. Algorithmically, that is, following the rules based on syllabification. However, many words cannot be hyphenated according to standard rules, and some may be hyphenated incorrectly by the algorithm.
2. By dictionary lookup. Words that are exceptions to rules of hyphenation have been stored in a file, and TEXTFORM looks up all words to be hyphenated in this file before it tries to hyphenate algorithmically.

To begin hyphenation, enter the command HYPHENATION ON, or HYPHEN ON, followed by one or both of ALGORITHM or DICT. If you want words hyphenated by either method (TEXTFORM hyphenates by algorithm when the word is not in the dictionary), the command is:

<HYPHENATION ON ALGORITHM DICT>

Throughout the rest of the document, HYPHEN OFF and HYPHEN ON can be used to stop or resume hyphenation. To prevent hyphenation around a specific word, be sure you have ended the word (with a blank) before turning hyphenation back on:

<HYPHEN OFF> specificword <HYPHEN ON>

Words containing ¬ (see page 29) or <BC> are not hyphenated.

If you want only algorithmic hyphenation, the command is:

<HYPHENATION ON ALG>

The algorithm attempts to hyphenate words according to English rules of syllabification. If your document is French, use the command:

<HYPHENATION ON ALG=FRENCH>

If you want all hyphenation to be done by dictionary look-up (to ensure total acccuracy), the command is:

<HYPHENATION ON DICT>

After this command, TEXTFORM hyphenates only those words that are in the file *TXTFHYPHDICT. If you edit this file, you will see that words are entered with hyphens indicating where the word can be broken. When a word overflows a line, and HYPHENATION ON DICT is in effect, TEXTFORM checks whether the word is in this file, and hyphenates accordingly.

You can add to this list of words by creating such a file yourself, for example, MYDICT, and then giving the command

<HYPHENATION ON DICT='ETC:TXTFHYPHDICT+MYDICT'>

In a hyphenation dictionary file, the words do not have to be in alphabetic order, and more than one word may appear on a line (up to 512 characters per line). This file may be edited to add or modify words. Upper and lower cases of the same word must be entered separately (TEXTFORM will not hyphenate 'UNIVERSITY' or 'University' if only 'university' is in the dictionary, in case you do not want the first word of a sentence to be hyphenated). However, TEXTFORM removes punctuation from the beginning and ending of a word before looking it up in a hyphenation dictionary. Lines beginning with '*' are comment lines.

```
# list mywords
> 1     al-ba-tross
> 2     com-pe-tent
> 3     to-geth-er
# End of file
```

If the word is a hyphenated word, such as 'pre-defined' or 'write-up', enter it in the hyphenation dictionary with two hyphens:

```
4     pre--defined
5     write--up
```

These words are broken only at the hyphen; in the above list the word 'pre-defined' would not hyphenate 'defined' unless the following appeared:

```
4     pre--de-fined
```

If you use automatic hyphenation, you can use the LIST HYPHENATION command (see Appendix 1) to produce a list of all words which were hyphenated by TEXTFORM. This lets you check which words have been hyphenated incorrectly; these should be added to your hyphenation dictionary file.

The shortest hyphenated word fragment contains MINHYPH or more letters (words must be at least twice the size of MINHYPH before hyphenation is attempted.) By default MINHYPH=2. If you do not want a word hyphenated unless at least 3 letters have appeared on the line, change MINHYPH:

<MINHYPH = 3>

If you have a large dictionary file, or a long TEXTFORM run, you can use an internal form of dictionary file to save costs. See page 201.

TEXTFORM LANGUAGE IN DETAIL

What is Computer Text Formatting?

Text formatting is the display of the written word—the arrangement and placement of text. Text is formatted in order to enhance the message and make it easier for the reader to understand what is being said.

Text can be formatted using a computer **text processing** program. TEXTFORM is a text processing program developed at The University of Alberta. It provides the user with the means of controlling all aspects of the format of a document.

TEXTFORM can produce text on several different devices. These include the IBM 1403 line printer, the Xerox 9700 page printer, the CalComp plotter, the Autologic APS5 and APSmicro5 phototypesetters, and the IBM 6670 printer.

Text Processing Concepts

Before you begin to learn more about TEXTFORM, you should become familiar with some of the basic text processing concepts and terminology.

As mentioned above, TEXTFORM is a text processing program. **Text** refers to the words that appear on the pages of your document. TEXTFORM distinguishes text from the formatting instructions (TEXTFORM **commands**) that control the way the text looks on the page but that do not themselves appear in the final product. Both text and TEXTFORM commands are entered into the input file (or **source file**), the file that TEXTFORM processes in order to produce the formatted document (the output file or print file). The machine on which your document is finally produced is called an **output device**. The default output device is the Xerox 9700 page printer, except at RPI, NCL and DUR universities, where it is the IBM 1403 line printer. TEXTFORM is often said to be **device independent** because you do not need to learn a different set of formatting commands for each device. Often, a simple change or two in the source file allows a document to be produced on another device.

In TEXTFORM, the term **page** means the size of page for which TEXTFORM prepares your document to be printed. On the page printer, for example, TEXTFORM assumes your page to be 8 1/2 x 11 inches in size. **Margins**, the white space around the page in which no text is printed, are considered to be part of the page. Within this page lies the text area, i.e., the area in which text will actually appear. The text area of an 8 1/2 x 11 inch page with four 1-inch margins is thus 6 1/2 x 9 inches in size. These dimensions can be changed, depending on the capabilities of the output device.

A **character** is a single element of text, i.e., a letter, number, or punctuation mark. A group of characters that are similar in design are called, collectively, a **character set**. Character sets usually consist of the entire range of alphabetic and

numeric characters plus a number of nonalphabetic and nonnumeric ones—punctuation marks, special symbols, etc. Within most character sets are a variety of type styles or **fonts**. Normal or Roman font characters are printed straight up and down; italics are *slanted* to the right; bold characters are **heavier** and **darker** than normal.

Character sets also have the quality of being **proportional** or **monospaced**. If proportional, each character has a different width. The letter “i”, for example, is thinner (i.e., takes up less room on the line) than the letter “o”. Uppercase characters are generally wider than their lowercase counterparts. Monospaced, or nonproportional characters, on the other hand, are all the same width. A document that is printed in a proportional character set takes fewer pages (usually about 1/3 fewer) to print than one that is printed in a monospaced character set. Most typewriters have monospace characters; most newspapers and books are prepared with proportional characters.

How Does TEXTFORM Work?

TEXTFORM processes text word by word, line by line, and page by page. A word is one or more nonblank characters followed by a blank, blanks, or the end of a line. (A word at the end of an input line does not need a blank after it.) The margins that are in effect tell TEXTFORM the length of a line on the output page. TEXTFORM puts as many words as it can on a line. When a word does not fit on the current line, TEXTFORM starts a new line. If this occurs at the bottom of a page, TEXTFORM starts a new page. TEXTFORM does not break a word across two lines unless the hyphenation facility is in effect. You can, of course, control line and page breaks yourself, with LINE and PAGE commands.

TEXTFORM has two main states or modes — text mode and command mode. When TEXTFORM encounters a < in text mode, it switches to command mode and expects to process one or more instructions, until it encounters a >.† Blanks around these characters are optional. If you want to have a < appear in the text, enter two in the input (<< produces <). If you want to have a > appear in the text, enter it as a normal character.

TEXTFORM instructions may be typed in any combination of uppercase and lowercase letters. Using the COMMENT command as an illustration, here are some ways in which TEXTFORM commands may appear in the source file:

```
one command <comment> in text
several commands <comment,comment, comment>together
a command right in the mid<comment>dle of a word.
< comment ><comment>
```

The characters > and , end the COMMENT command. If two commands appear in succession, they may be entered within the same < and > by separating them with a comma, the *command separator*. Blanks on either side of the comma are optional. When a command has keywords, the keywords are separated by blank spaces, and not commas. The comma is used between distinct commands, not parts of the same

----

†However, if the < or > appear in an IF command (see page 156) they do not indicate the start or end of command mode.

command.

If you want to continue commands onto the next line, you must use the – (hyphen or *continuation character*) which is the character on your keyboard:

```
<COMMENT, COMMENT, –
COMMENT, COMMENT this is a long –
comment >
```

The continuation character must be the last character on the line; no blanks may follow it.† (Hyphens elsewhere in the text are treated as normal text characters.)

If you fail to end a command line with the command terminator or continuation character, TEXTFORM prints an error message *Missing continuation line character*, and then switches to text mode. The result of this could be undesirable since TEXTFORM will now process the following line of commands as text.

To summarize, the following characters have special uses in command mode:

command terminator >
continue line –
command separator ,

Items That Can Appear in Command Mode

Within command mode, TEXTFORM recognizes only certain types of items. Any unrecognizable items cause error messages.

Commands have an immediate effect—for example they begin a line or paragraph. LINE, NEWPARA, and PAGE are commands. Most commands have abbreviations, because they are typed frequently. Many commands correspond to an action at a typewriter. The NEWPARA command is like hitting return and spacing in five spaces at the typewriter.

Other items have been described which contain information, such as LINESPACE and ALIGNMENT. These are called **variables**, and are values that TEXTFORM uses periodically. You can recognize a variable by the equal sign (=) that is used to give a new value to the variable, as in LINESPACE=8MM. On a typewriter, a mechanical setting such as the line spacing or tab stops achieves the same result as variables do in TEXTFORM.

Although commands and variables have been distinguished here, throughout the manual you will find that most sequences of instructions within < and > are called *TEXTFORM commands*, even though they may contain variables. All the commands and variables are listed in Appendix 1, along with other items that will be referred to in this manual.

Another type of item that appears in command mode is a **string**, used when it is necessary to supply text while in command mode. A string is enclosed within matching **delimiters**, either ' (single quotes) or " (double quotes):

----
†At NCL and DUR universities it must be the last non-blank character on the line.

```
<comment, 'text text', comment>
```

produces:

```
text text
```

Within a string, TEXTFORM is once again in text mode. (TEXTFORM cannot be both in command and text mode simultaneously.) If the text of the string itself contains a quote, you can use the alternate delimiter, or double the quote where it appears in the text:

```
<comment, 'That''s it!'>
<comment, "That's it!">
```

both produce:

```
That's it!
```

A continuation character cannot be used to continue a string over two lines. Instead, put a delimiter at the end of the first part of the string, followed by a : (colon) (the catenation operator) and a continuation character (–). Then put the second half of the string, within delimiters, on the next line. For example:

```
< 'This is a string ': –
  'which spans two lines' >
```

is equivalent to:

```
< 'This is a string which spans two lines' >
```

The continuation character can also be used in text mode, where it prevents a blank from being inserted at the end of a line:

```
a word kept toget–
her over two lines
```

produces:

```
a word kept together over two lines
```

However, it is simpler to start a new word on a new line than to use the continuation character in text.

Several forms of TEXTFORM instructions expect information to be supplied. This information, called a **parameter**, appears within matched parentheses:

```
<COMMAND( 'This is the information' )>
```

If the information is a string, it can be broken across two lines:

```
<COMMAND('This is a string ': –
  'which spans two lines')>
```

is equivalent to:

<COMMAND('This is a string which spans two lines')>

Characters Which Have Special Uses in Text Mode

Text mode is TEXTFORM's normal mode. Most text characters in the input appear in the output. However, there are five characters which perform special functions when encountered in text mode. They are called **meta-characters** because rather than being a normal text character, they cause a special action on what follows. The meta-characters are:

non-line-breaking word space ¬
emphasize next character _
capitalize next character @
command initiator <
continue line –

Non-Line-Breaking Character

The *not sign* ¬ provides a non-line-breaking word space in text mode. It may be inserted instead of a blank. During formatting the non-line-breaking character is replaced by a word space, but words on either side of the character are forced to appear on the same output line. If there is not enough room on the current line, the words will appear on the next line:

Dr.¬W.¬A.¬Gonzo, alias Huntress¬Thompson
Address:¬13¬Willow¬Valley¬Mountain¬Top
Phone:¬432-1313

produces:

Dr. W. A. Gonzo, alias Huntress Thompson
Address: 13 Willow Valley Mountain Top Phone: 432-1313

To force the *not sign* to be printed in text, repeat it once for each one required:

2+2 ¬¬= 5

produces:

2+2 ¬= 5

To reduce editing, insert this character between phrases such as 'et¬al', '30¬May', or 'J.¬O.¬Bloe' when you type the text into a file. This ensures that these phrases are not broken across two lines.

The character simply ensures that words appear on the same line. It does not determine the amount of space that appears between the words. If the text is being justified by the program, TEXTFORM may insert justification space where the non-line-breaking character appears. If a justification space is not desired, use the BLANKCHARACTER command.

The non-line-breaking character may not have a blank on either side of it. For example:

et¬ al

produces the error *Non-line-breaking word space specified incorrectly. It is ignored.*

Emphasize Next Character

The *underscore* character _ can be used to emphasize the character following it, as described on page 38:

_T_E_X_T_F_O_R_M

produces:

TEXTFORM

The underscore must be entered twice if you want one in the output:

__H__i

produces:

_H_i

Capitalize Next Character

Upper case letters are normally entered using the shift key at the terminal. However, the *at sign* @ may be used to force the next letter to be converted to upper case.† Two *at signs* together produce one *at sign* in the output.

@text@@

produces:

Text@

If there is no capital for the next character, a *Character is not available. A blank is provided as replacement* message may be generated, and the capitalization process ignored.

----

†This facility is seldom used, unless you need to force the first character of a word to appear in upper case, as on page 98. See CAP, UPPERCASEINPUT, UC, and LC in Appendix 1 for other ways to control whether characters are upper or lower case.

Command Initiator

The command initiator < tells TEXTFORM to switch into command mode. If you need to produce the < character in a document, enter two for each one that you require. This can only be done from text mode. You cannot enter << while in command mode to produce that character in the output.

The > character has no special use in text mode, so it is not doubled. In command mode, it indicates the end of the command.

Continue Line Character

In text, the character – is a normal text character, unless it is the *last* character on a line. In this case, it prevents the end of a word, and thus prevents a word space from being inserted. If it is followed by a blank, or if it appears in a hyphenated word, such as write-up, it is a normal character.

Lengths in TEXTFORM

Lengths are numbers with associated units of measurement. For example, LINESPACE=.33INCH gives the length .33 inches to the variable LINESPACE. When TEXTFORM requires a length, the length can be given in any of the following units. Notice that there must be no blanks between the number and the unit of measurement—the correct form is 9IN and not 9 IN.

INCH, INCHES or IN
MILLIMETER, MILLIMETRE, MILLIMETERS, MILLIMETRES or MM
PICA, PICAS or PI
POINT, POINTS or PO
UNIT, UNITS or UN
LINE, LINES or LI

When a length is given in a horizontal or vertical direction, the actual length produced is the closest the output device and character set can produce. For example, if you request a LINESPACE of .2 inch, it may actually be .16 inch on some output devices, although the variable still contains .2 inch.

If a length is expected, but no unit of length is indicated, a **default unit** of length is used. This default units value is stored in DEFUNITS. At the start of a run, DEFUNITS contains MILLIMETERS, so the command LINESPACE=4 is treated as LINESPACE=4MILLIMETERS. DEFUNITS can be changed to contain other units of length:

<defunits = inches>
<linespace = .16>

More information on lengths is given on page 100.

Notation Used For Subsequent Parts of This Manual

The following notation is used in this manual to define TEXTFORM instructions, and is used in Appendix 1. Italics indicate messages and information displayed by TEXTFORM during the run.

[ ]

When [and] appear, what they enclose is optional.

. . .

indicates that the last item may be repeated as many times as necessary (e.g. 'kwd . . .' means that you may include any number of the keywords, each separated by one or more blank spaces).

body

is any valid TEXTFORM input. It may be a number of commands, some text, or both. It need not all be on the same input line.

c

single character string.

CAPITAL letters

Letters which are **CAP**ITALIZED must appear (although you may enter them in upper or lower case). The characters which appear in bold are an allowable abbreviation.

expression

any expression containing valid variable names (subscripting and substringing is allowed), and/or valid function calls, and/or valid literals; separated by operators. A complete description of expressions is found on page 112. If a command takes an expression, as in <UNDERLINE expression>, 'expression' may be a string variable:

```
<DEF  &ON='ON'>
<UNDERLINE  &ON>
```

kwd

must be one of the keywords described following it. The keywords are capitalized in the description and are separated by blanks. If a keyword in a command is preceded by ¬ or – the effect is to negate the action of the keyword.

If a command takes a keyword, and the keyword is in a string variable, as in DEFINE &T='TRY', the variable must be executed (see page 108) if it is used in a command:

```
<HYPHENATION  ON  $&T>
```

length

is any valid expression. If the result of the expression is not a length, the default length units (in DEFUNITS) is used to convert the result to a length.

logical value

either TRUE, (ON, YES are synonyms), or FALSE, (OFF, NO are synonyms). An expression which returns the value 1 (TRUE), or 0 (FALSE) may also be used.

name

a valid name. Predefined names start with a letter (a-z). A user-defined name must start with an &, and may contain any of the characters a-z, _, &, or 0-9. The characters in the name may be in upper case, lower case, or both; so the names &ABC and &Abc are the same. The name may be of any length.

name = expression

is the form of an assignment. ‘name’ may simply be a name, or a subscripted name (which will change the structure element of ‘name’ which has the specified subscript). ‘name’ may also refer to a substring, in which case a substring of ‘name’ will be changed.

number

is any valid expression which results in a number (or a string which can be converted to a number). If the result of the expression is a length, DEFUNITS is used to convert it to a number. If the result of the expression is a scaled number, it is rounded (in most cases — note that ‘number’ in some cases may be scaled).

par

a function parameter. Each parameter may normally be an expression; however in cases where the function parameter RETURNS something or TAKES a NAME, the parameter must be the name of a variable. See the LOAD command on page 210 for a description of TAKES, NAME and RETURNS.

string

any characters between paired quotes (either " or ').

structure

A structure is either the name of a variable containing a structure, or (expression, expression2, . . ., expression_n)

User-Defined Names in TEXTFORM

Much of the rest of this manual deals with user-defined items. This section gives information about user-defined names.

Before a name is used, it must be **defined** so that TEXTFORM can add it to the list of of all names that it recognizes. You can define a name one time only. If you try to define the same name twice, the second definition is ignored and you get the error message name *is already defined.* However, using a name that is not defined causes the error *'name' is not defined. Ignored.*

The DEFINE command defines a name:

<**DEF**INE [kwd] name >

kwd
may be given to indicate the type of cross referencing for ‘name’. Cross references are described on page 176.

name
- must start with an & (ampersand)
- may contain the characters of the alphabet (a–z), the 10 digits (0–9), and the character & (ampersand) and _ (underscore). No other characters are allowed.
- may be of any length up to 256 characters
- All names are converted to upper case by TEXTFORM, so &IND is the same as &ind.

Choose an Output Device for the Document

The output device is where the final copy of a document appears. It may be a terminal, printer, phototypesetter, or plotter.

At the beginning of a TEXTFORM run, before any text is formatted, an output device should be chosen. If you do not do this, TEXTFORM uses the default, which is the Xerox 9700 page printer (at RPI, NCL and DUR universities, the default is the IBM 1403 line printer). The command to specify the device is:

<**O**UTPUT**D**EVICE [expression [expression2 [expression3 [expression4]]]] >

There must be a space between the parts of this command. It is usually used as follows:

<outputdevice 'x9700' 'univers'>

‘X9700’ represents the output device, the Xerox 9700 page printer. ‘UNIVERS’ is one of the available character sets on this device. ‘expression3’ and ‘expression4’ are described in Appendix 1.

Each output device may have several sets of characters. On some devices, only one set can be used in a single TEXTFORM run. Instructions on using the various devices may be found in CC Memos. Appropriate documentation for each device is

also listed.

Once the OUTPUTDEVICE command has been given, the part of the program that deals with that device becomes **loaded**. Once the output device has been loaded, it cannot be changed for the rest of the run. If you do not give an OUTPUTDEVICE command, TEXTFORM still loads the default output device when it encounters text, or when it encounters a command containing a length or a command that uses a length.

Getting a Proof or Working Copy of a Document

At the time of publication, the facilities described in this section were not properly implemented at U of M. When implementation has been completed, a full writeup will be issued.

TEXTFORM can produce the formatted text in a manner which eases the process of making corrections and changes. If the PROOFDEVICE command appears, the formatted copy is produced with source line numbers running down the left hand side, just as you saw on the listing of the input file. These line numbers make it easy to find the appropriate line in the input file for changes or additions.

The PROOFDEVICE command is similar to the OUTPUTDEVICE command:

<**P**ROOF**D**EVICE [expression [expression2]] >

It must be used *after* the OUTPUTDEVICE command.

<od 'x9700' 'tn'>
<pd 'x9700' 'tn'>

When the PROOFDEVICE command appears, the proofed text and the statistics are produced on SPRINT. The command to run TEXTFORM becomes:

run *textform scards=*file* sprint=*result*

Although proofs are often used for working copies, they are especially useful in cases where the output device is expensive, has poor turnaround, or is slow, which may be the case with phototypesetters. When used for such a device, the proof device attempts to approximate the appearance of the output page. If it has fewer capabilities (for example, the printers cannot change typesize) than the intended output device, messages are printed indicating the action which will take place on the output device.

The following table lists the devices which can be used as proof devices. Not all output devices have proof devices. When the proof device is the same as the output device, specify SPRINT instead of SPUNCH in the RUN command.

device	produces proofs for	using characterset
1403	1403	same as odcharacterset
	APS5	TITAN10TN
X9700	X9700	same as odcharacterset
	APS5	TN
		TITAN10TN
RASTER	APS5	DEFAULT

Sample Proof

```
# list file

>  1    <OD 'X9700' 'TITAN10TN', PD 'X9700' 'TITAN10TN'>
>  2
>  3    <font bold>
>  4    Quotations from Gulliver's Travels
>  5    <lineend, vertspace linespace, newpara, font normal>
>  6    And he gave if for his opinion, that
>  7    whoever could make two ears of corn or two blades
>  8    of grass to grow upon a spot of ground where only one
>  9    grew before, would deserve better of mankind,
> 10    and do more essential service to his country,
> 11    than the whole race of politicians put together.
> 12
> 13    <newpara>
> 14    He had been eight years upon a project for extracting
> 15    sunbeams out of cucumbers, which were to be put in vials
> 16    hermetically sealed, and let out to warm the air in raw
> 17    inclement summers.
> 18
> 19    <lineend, vertspace linespace> Jonathan Swift
# End of file

# run *textform scards=file sprint=–result
# 14:23:16     RC=0
# copy –result *sink*
```

```
     4      Quotations from Gulliver's Travels

     6          And he gave if for his opinion, that whoever could make
     7      two ears of corn or two blades of grass to grow upon a spot
     8      of ground where only one grew before, would deserve better of
    10      mankind, and do more essential service to his country,
    11      than the whole race of politicians put together.

    14          He had been eight years upon a project for extracting
    15      sunbeams out of cucumbers, which were to be put in vials
    16      hermetically sealed, and let out to warm the air in raw
    17      inclement summers.

    19      Jonathan Swift
```

CHARACTER APPEARANCE

Underlining

Underlining is controlled with the UNDERLINE command.

<**U**NDERLINE [expression]>

UNDERLINE ON, or UNDERLINE, or U, begins underlining; UNDERLINE OFF stops it.

Underlining <U> is done in this manner: <BC 1IN><U OFF>.

produces:

Underlining is done in this manner: ______________.

The effect of this command can be modified in several ways. The variable UNDERLINEWORDSPACE determines whether word spaces are underlined. By default, it is FALSE. If you want word spaces to be underlined, set it TRUE:

<UNDERLINEWORDSPACE=TRUE>Underlining <U> is done in this manner: <BC 1IN><U OFF>.

produces:

Underlining is done in this manner: ______________.

The variable UNDERLINESTRING contains the character that is used for underlining. At the start of the run, it is '_'. It can be changed on some devices (see the CC Memo for each device for details), as in the following example:

<UNDERLINESTRING = LIGHTRL,U ON>lightrule<U OFF>

produces:

lightrule

CURUNDERLINE also gives information about underlining. It is 1 when underlining is on, and 0 when underlining is off. For example, to test whether an UNDERLINE command has been given, use:

<IF CURUNDERLINE = 1, THEN, . . . >

The variable UNDERLINEDISPLACEMENT can be used on APS5 and CALCOMP output devices to change the vertical position of the underline character, which touches the bottom of characters. It can be changed to a positive or negative length. To place UNDERLINESTRING 1 point lower:

<UNDERLINEDISPLACEMENT = 1POINT>

Fonts

In many instances, fonts which change the appearance of the character are used to emphasize text, delimit text, or to indicate special meaning. There are different fonts for each character set, but most character sets have the same basic fonts. Fonts are specified by the FONT command:

<**F**ONT [expression]>

The commands FONT 1, FONT, or F, return to the normal font. Valid fonts are:

<FONT 1> or <FONT ROMAN> is normal text
<FONT 2> or <FONT ITALIC> *is italic text*
<FONT 3> or <FONT BOLD> **is bold text**

Some character sets have four or more fonts:

FONT 4 is ***BOLDITALIC***
FONT 5 is EXTRABOLD
FONT 6 is EXTRABOLDITALIC
FONT 7 is LIGHT
FONT 8 is LIGHTITALIC

Emphasis can also be accomplished without specifying the actual font desired. The *emphasize next character*, the underscore _ , may be used to do this:

_e_m_p_h_a_s_i_s

produces:

emphasis

Depending on the current font, the emphasis character uses a emphasis font that will emphasize the next character. Not all fonts have an emphasis font in which case the error *Emphasis is not available in the current FONT. Emphasis ignored* is produced. The font chosen depends on the capabilities of the output device. This is indicated in the individual device write-ups. For example:

<FONT 1>
this is a _t_e_s_t for emphasis
<FONT 2>
this is a _t_e_s_t for emphasis

produces:

this is a *test* for emphasis *this is a* test *for emphasis*

CURFONT and CUREMPFONT provide information about the current font. TEXTFORM puts the font number in CURFONT whenever you give a FONT command. This lets you get the current font number, change fonts, and return to the

original font without even knowing what that original font was:

<define &save = curfont, font 3> Title <font &save>

CUREMPFONT is the number of the emphasis font for the current font. If there is no emphasis font, it contains 0.

Typeface or Character Set

Many output devices have several typefaces, or character sets. This is specified in the OUTPUTDEVICE command, after the output device name. In the following example TN is the character set.

<OUTPUTDEVICE '1403' 'TN'>

Some devices permit you to change typefaces during one TEXTFORM run by using the CHARACTERSET command which has the form:

<**C**HARACTER**S**ET string>

The character set name is a string, enclosed in delimiters, and is hyphenated if it is two words. It cannot contain blanks. The CHARACTERSET command is used after the OUTPUTDEVICE command, not as the first command in a run. For example, in the Postscript output, you might begin with the CENTURY-SCHOOLBOOK character set:

<OUTPUTDEVICE 'POSTSCRIPT' 'CENTURY-SCHOOLBOOK'>

but later need a fixed-pitch font for an example:

The statement <CS 'COURIER'>READ(5,101) XVEC, YVEC<CS 'CENTURY-SCHOOLBOOK'>reads a . . .

produces:

The statement `READ(5,101) XVEC, YVEC` reads a . . .

The variables CURCS and ODCHARACTERSET are related to this command. CURCS is an upper case string containing the current character set name. It is changed by TEXTFORM whenever you give a CHARACTERSET command. For example:

<IF CURCS = 'GREEK', THEN, . . . >

ODCHARACTERSET is an uppercase string containing the character set name that was given in the OUTPUTDEVICE command. This character set is used at the start of all footnotes (see the discussion of default state on page 55).

Producing Special Characters

Many character sets contain characters that are not available on the terminal keyboard. To produce these characters in the output document, enter the name of the character as a command. For example, produce a superscript one or an umlaut or an integral with the commands:

```
<sup1>    <umlaut>    <integral>
```

No one character set contains all special characters. Consult the documentation for each of the output devices to see which characters are available in each character set.

If the requested character is not available in the character set being used, an error message is produced: *Character is not available. A blank is provided as replacement.* You can often prevent this error by using CHAREXIST to check whether the character exists in the current character set before using it. For example, if you require a copyright symbol, but are willing to use the letter 'C' if there is no copyright symbol:

```
<if charexist(copyright) = true, then, copyright,  else>C<endif>
```

Some characters are not available in all fonts. When TEXTFORM replaces the character with one from another font, it produces a warning. For example, if the = is not available in FONT 2, the commands:

```
<OUTPUTDEVICE 'APS5'>
<FONT 2>4 + 4 = 8
```

produce the warning *Character is not available in the current font. A replacement is provided by an alternate font.* This is not an error message. It is merely warning that the character may have a different appearance.

If you are using a large number of special character names from the APL character set, you may be able to reduce typing by using the KEYBOARD command, described in Appendix 1.

Superscripts and Subscripts

Some character sets have a limited number of **super** or **subscript** characters (characters which are positioned slightly above or below the normal text characters). These can be produced by names such as <SUP5> and <SUB3>. The characters are available *only if they appear in the write-up for the character set*.

```
C<sup1,sup2>
```

produces:

C^{12}

When a device can space vertically in amounts less than LINESPACE, it may be possible to generate super and subscript characters by issuing vertical space commands and reducing the typesize (if the device can do it).

<define &supa="<bc 0, vs –tsize/2, 'A', vs tsize/2>">

These commands shift an A up half the current typesize. The BC command ensures that the word is started in the correct position on the line, before the negative vertical space.

When VS is used to produce subscripts, use KEEP and ENDKEEP so that the VS command does not overflow the page in the middle of a line.

<define macro &SUB,
  bc 0, vs tsize/2, keep, par(1), vs –tsize/2, endkeep, –
edef macro &SUB>

Overstruck Characters

Characters can be overstruck by using the LOGICALBACKSPACE command.

c<**L**OGICAL**B**ACK**S**PACE>c

This is how accents are positioned over letters, if the current character set contains accents.

u<lbs, umlaut>

produces ü. When this command is used, the character which follows is backspaced and centred over the previous character. Commands to change fonts or character sets can appear between the LBS command and the characters.

<EMDASH, LOGICALBACKSPACE, FONT 3>|<FONT 1>

produces:

+

Neither character around the LBS command may be a blank; this produces an error.

abc <LBS>d

produces the error *Logical backspacing used incorrectly. Ignored.* and the output:

abc d

If a BC command appears immediately before or after an LBS command, *it* is the 'character' that is used in the LBS sequence. When working with the meta-characters, remember to double them:

__<LBS>@@   <'@@', LBS, '__'>

both produce:

@

Overstriking Characters without Centring

Negative horizontal movements can be used to overstrike characters, without centring the characters on one another. For example, overstrike a vertical bar one third of the way from the previous character:

```
–<BC  –TEXTWIDTH('–')/3,  VBAR>
```

produces –| if the output device can move a distance less than the width of a character.

Automatically Replacing Text Characters in the Input

The AT TEXTCHARACTER, or AT TC command, lets you include a TEXTFORM name instead of a character whenever that character appears in the text. For example,

```
<DEFINE  &BREAK  =  '/<ALLOWLINEBREAK>'  >
<AT  TEXTCHARACTER  '/'  &BREAK>
```

inserts &BREAK, which is ‘/<allowlinebreak>’, whenever a ‘/’ appears in the input text. It is not in effect in command mode. While &BREAK is being included, the ATs for the ‘/’ character are disabled, although ATs for all other characters are still in effect. The command can also be used with special character names, as in

```
<AT  TC  INTEGRAL  &INT>
```

The AT command could be used to ‘turn off’ meta-characters. If you want the @ to be a normal character, use:

```
<DEFINE  &ATCHAR  =  '@@'  >
<AT  TC  '@'  &ATCHAR>
```

Two characters are special cases when this command is used.

1. If you specify AT TC ‘<’, the AT is done only when *two* command initiators appear in the input. It is not done for a single command initiator.
2. If you specify AT TC ‘–’, the AT is not done when the ‘–’ is used as a continuation line character. However, it is done elsewhere on the line.

More than one AT can be given for a character. The ATs are done in a last-in first-out order, so that the most recently specified AT for a character is done first. Characters with associated ATs can be used in PREFORMATTED mode, but must not be used in ASIS mode, because TEXTFORM does not recognize the commands necessary to do the AT during ASIS mode. If KEYBOARD is APL1 or APL2, ATs are not in effect.

During a table of contents or index entry, the AT remains in effect, and the contents of the name, rather than the original text character, are stored. However, when a SPLIT command is used, as in SPLIT '.', the ATs for the split string character are not done.

To stop an AT for a character, use the SUSPEND AT TEXTCHARACTER 'x', or SUS AT TC 'x' command. For example, to suspend all the ATs for the character '*' use:

<SUSPEND AT TEXTCHARACTER '*'>

To suspend a specific AT, e.g. the one with the name &REPLACE for the character LEFTA, use:

<SUS AT TC LEFTA &REPLACE>

AT_INFO, as shown in Appendix 1, tells what AT names are associated with a text character.

Typesize

Some devices let you change the size of characters. This is known as **typesize**. If your device does not have this capability, this section will not apply to you, because any attempt to change typesize produces the warning *TYPESIZE command not allowed for this output device. Command ignored.*

To change the typesize, issue the command:

<**TYP**ESIZE length>

On devices that can change typesize, the default typesize is 10 points (**points** are a typesetting unit of length—there about 72 points in an inch). To change the character size to 24 points, enter the following command:

<TYPESIZE 24POINTS>

Since typesize is actually a measurement, you can express it by using any of the length measurements accepted by TEXTFORM, although points are most commonly used. If the device cannot produce the exact size that you request, TEXTFORM produces the closest possible value. The device write-ups describe the capabilities of each device.

Although typesize refers to the vertical size of the characters, it cannot be precisely measured with a ruler. All the different type styles of 12 point characters, for instance, are approximately the same size, but there is some variation. Only with considerable experience will you be able to determine the typesize of a character. The closest test is to measure the line spacing by measuring the vertical distance between **baselines** (the imaginary line on which the text seems to rest). Typesize is usually one or more points smaller than the line spacing.

If you change typesize, you will usually adjust the value of LINESPACE to be several points larger than the typesize. WORDSPACE is usually adjusted at the same time.

During a TEXTFORM run, TSIZE contains the typesize. It is changed by TEXTFORM whenever you give a TYPESIZE command. For example:

```
<define &oldtsize = tsize>
<typesize 8point, linespace = tsize + 2point>
text
<typesize &oldtsize, linespace = tsize + 2point>
```

Other Modifications to Text Appearance

ALPHABETIC converts integer to an alphabetic counter.

```
<ALPHABETIC(2)>  <ALPHABETIC(28)>  <ALPHABETIC(26*2+10)>
```

produces:

b ab bj

ENGLISH converts number to lowercase cardinal English.

```
<ENGLISH(17)>
```

produces:

seventeen

FRENCH converts an integer number, to lowercase French text.

```
<FRENCH(68)>
```

produces:

soixante-huit

ROMAN converts a number to roman numbers.

```
<ROMAN(3)>
```

produces:

iii

LOWERCASE converts text to lower case.

```
<LOWERCASE('TITLE  1')>
```

produces:

title 1

UPPERCASE converts text to upper case.

```
<UPPERCASE('parameter')>
```

produces:

PARAMETER

These techniques of modifying text appearance can be used together. For example, to convert a number to an uppercase roman number:

<UPPERCASE(ROMAN(3))>

produces:

III

PAGE APPEARANCE

Size of Page

The default page size is 8.5 inches by 11 inches. The page size can be changed by a command of the form:

<PAGESIZE=(horizontal length, vertical length)>

for instance:

<PAGESIZE=(7.5IN,11IN)>

makes TEXTFORM format text for a page 7.5 inches wide and 11 inches long.

There are also values which can be used to produce metric page sizes. See A0–A8 in Appendix 1.

The size of the page must conform to the confines of the output device being used. The line printer (IBM 1403) has a maximum of 13.2 inches by 11 inches, the page printer has a maximum of 8.5 inches by 11 inches (or 11 inches by 8.5 inches) and the APSmicro5 has a maximum of 11.67 inches by 45 inches.

A change of page size does not take effect until the next page is begun. Either change the page size at the start of the document, before any text is given, or be sure to begin a new page after the change.

Once a page has been started, TEXTFORM stores the size of the page *currently being formatted* in CURPAGESIZE; if you change PAGESIZE, it says what size to make the next page.

Margins

The area in which text is printed is determined after the four margins are subtracted from the page size. The four page margins are called:

TOPMARGIN
BOTMARGIN
LEFTMARGIN
RIGHTMARGIN

Their default values are all 1 inch. Change LEFTMARGIN with:

<LEFTMARGIN=1.5IN>

As with PAGESIZE, changes to the margins take place only when TEXTFORM begins a new page, even though you may have entered the command in the middle of

the previous page. The INDENT command, which has already been discussed, can be used to temporarily modify the left and right margins.

Once a page has been started, CURLEFTMARGIN, CURRIGHTMARGIN, CURTOPMARGIN, and CURBOTMARGIN contain the lengths for the page currently being formatted. Any changes to LEFTMARGIN, RIGHTMARGIN, TOPMARGIN and BOTMARGIN specify the margins for the next page.

Refer to the section Front and Back Pages if you want to make the margins different for front and back pages.

Page Position Commands

When the current page is full, TEXTFORM starts the next page. However, the command

<**P**AGE**END**>

forces the immediate end of the page, if one has been started. Page size and margin changes can then be made before starting the next page. For example:

```
<PAGEEND>
<PAGESIZE = (11IN,8.5IN)>
This text is on the next page.
```

An alternate page positioning command is PAGE:

<**P**AGE>

This command actually starts the new page, using the current value of PAGESIZE and margins.

If the text is being produced with more than one column per page (see page 78), the COLUMN command starts the next column. If used on the last column of the page, this command has the same result as the PAGE command.

Blank Pages

If you are currently at the end of a page, and no text has been placed on the page, PAGEEND does not force a blank page. The commands:

<PAGE, PAGEEND>

are needed to create a blank page. When you want the *next* page to be left empty, without issuing a PAGEEND command, use the following command, described in detail starting on page 60.

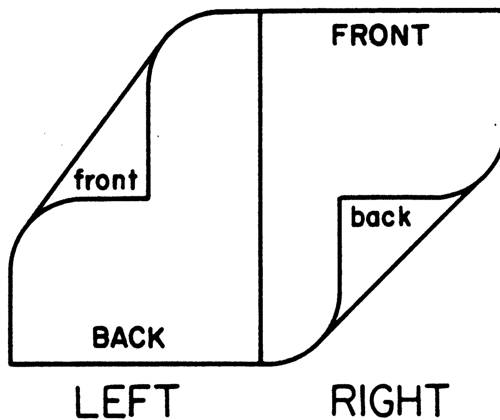
<FLOAT VERY TOP PAGE PAGEEND>

Front and Back Pages

Pages can be referred to as they would appear in a book. There are LEFT (or BACK) pages and RIGHT (or FRONT) pages. Pages with odd numbers are RIGHT pages while even numbered pages are LEFT pages.

In a TEXTFORM run, THISPAGE contains LEFT or RIGHT to indicate which page is currently being formatted. You do not change this variable—TEXTFORM changes it at the end of each page, immediately after a page overflows or after a PAGEEND command. TEXTFORM determines the value for THISPAGE by checking PNCTR, the page number counter. If PNCTR is an odd number, THISPAGE is set to RIGHT; if PNCTR is an even number, THISPAGE is set to LEFT. If you want text to start on a RIGHT (or FRONT) page use the following test:

```
<pageend>
<if thispage = left, then, page, pageend, endif>
<comment now start a front page>
```



The variable PRINTON allows TEXTFORM more control over devices that can print on both sides of the paper. It defaults to PRINTON=BOTH, and can be BOTH, RIGHT, LEFT, FRONT, or BACK. The setting of PRINTON affects how THISPAGE is changed.

If PRINTON has a value of BOTH, then THISPAGE is cycled between LEFT and RIGHT. In addition the choice of printing side is determined by the value of the variable PNCTR. If PNCTR is odd, then this is a RIGHT page; an even PNCTR implies a LEFT page.

If PRINTON is RIGHT or FRONT, then THISPAGE is always RIGHT. Likewise, a PRINTON setting of LEFT or BACK makes THISPAGE always LEFT. Note that when PRINTON is anything other than BOTH, THISPAGE and PNCTR are not logically connected.

Special care should be taken when setting PRINTON to anything but BOTH. Other features of the TEXTFORM language, like FLOAT or RESERVE, allow items to be reserved for a particular side of the page. If PRINTON=FRONT at the end of a run, the floats and reserves specifically for BACK pages will never appear.

When the **inside margin** (the one closest to the binding) must be larger to allow space for binding, the following instructions in your file will cause TEXTFORM to change the margins appropriately if your document is being printed on front and back pages.†

```
<define &INMARGIN=2IN, define &OUTMARGIN=1IN>
<leftmargin = &inmargin, rightmargin = &outmargin>
<def &bf = '<leftmargin=&outmargin, rightmargin=&inmargin>'>
<def &bb = '<leftmargin=&inmargin, rightmargin=&outmargin>'>
<comment to cycle margins>
<reserve bottom front page size default 0in &bf>
<reserve bottom back page size default 0in &bb>
```

Page Numbers

A TEXTFORM run starts on page 1, with THISPAGE=RIGHT. When a page ends (due to text overflowing the page or a PAGEEND command) the value of THISPAGE changes, and PNCTR, the variable containing the page number, is increased by one. PNCTR can be changed at any point, and is changed immediately:

```
<PAGEEND, PNCTR=5>
<COMMENT now start page 5>
```

or

```
<PNCTR = PNCTR + 3>
```

Although PNCTR is always incremented, it is not automatically printed. Page numbers are produced with the RESERVE command, described in detail on page 63. To cause the page number to be printed on the upper right of each page, do the following once:

```
<define &ur = '<pnctr, lineend right>'>
<topmargin = .5in>
<reserve top size 6.5in .5in &ur>
```

To start page numbers at 1 after a title page:

```
Title page <pageend, pnctr = 1>
<define &ur = '<pnctr, lineend right>'>
<topmargin = .5in>
<reserve top size 6.5in .5in &ur>
```

----

†If you are using a logical page, described later in the manual, issue the RESERVE commands after you USE the logical page.

Keeping Text on the Same Page or Column

The KEEP and ENDKEEP commands ensure that text which appears between the commands appears in the same output column.

<**KE**EP . . . **E**ND**KE**EP>

These commands can be used to prevent **widows** (single lines alone at the bottom of a column) and **orphans** (words appearing alone at the top of a column).

If a KEEP is used, and the end of the column occurs before an ENDKEEP command is encountered, TEXTFORM moves the text that it has formatted since the KEEP to the top of the next column, and then continues processing text as normal. If a command to end the page occurs within a keep, it produces the warning *A NEWPAGE or PAGEEND command given within a KEEP. The KEEP is stopped.* A similar warning appears if a column ending command appears.

KEEP can be used anywhere on the line. When the command is used, the boundary of the keep starts *at the beginning* of the line TEXTFORM is formatting. As a result, all of the following text is kept together, which probably isn't what you want:

```
. . .the end of this sentence.
<KEEP>
<LINE> This keep includes the previous line.
<ENDKEEP>
```

The previous example should be changed to:

```
. . .the end of this sentence.
<LINE, KEEP> This keep does not include the previous line.
<ENDKEEP>
```

Note that KEEP and ENDKEEP do not affect TEXTFORM's line ending decisions. They merely start 'keeping' text from the current *formatted* line (rather than *input* line) until the end of the keep. If a keep was previously in effect on the current formatted line, the two keep's are treated as one, as in this example:

```
<KEEP> This text will all be treated as
one 'keep'.
<ENDKEEP>
<KEEP>
<LINE> This is still part of the same keep. <ENDKEEP>
```

To treat the text of the previous example as two keeps, it should be:

```
<KEEP> This text will all be treated as
one 'keep'.
<ENDKEEP>
<LINE>
<KEEP> This is a new keep since the KEEP command is given on a
new formatted line. <ENDKEEP>
```

KEEPs are also 'nestable': the commands KEEP . . . KEEP . . . ENDKEEP . . .

ENDKEEP . . . are treated as one KEEP.

KEEPs cannot be used within tables, and should not be used around tables. KEEPs may not be used in footnotes or around text which contains FOOTNOTE and ENDFOOTNOTE. This will cause the error *FLOAT, FOOTNOTE, or KEEP is already being built. You can't build one inside another, sorry.*

As described above, when the text within the keep is taken to a column on a new page, the text appears *exactly* as it would have appeared on the previous page. For example, if the keep starts with:

```
<KEEP>This is page <PNCTR>.
```

then TEXTFORM will place the current value of PNCTR on the formatted line. However, if the keep is subsequently carried to the next page, the text will contain the value of PNCTR for the previous page. If you are merely storing the value of PNCTR, one way to minimize the problem is to do it immediately before the ENDKEEP command, because by that point TEXTFORM will have determined whether the keep will fit or be carried over.

Vertical Position of Text

Extra Vertical Space

The commands VERTSPACE, or VS, and VERTGAP, or VG, insert extra vertical space on a page or column of a page.

VERTSPACE spaces down immediately by the length specified, without changing the horizontal position. If the length is negative, it spaces back up the column. On the first line of a column, you can space vertically upwards by any amount less than LINESPACE. You cannot space back into the margins, although from a bottom reserve you can space upwards to print in a top reserve.

When a VERTSPACE command overflows a column, and it is not part of a word (for instance, after a blank), only the available length left in the column is used: the rest does not appear at the top of the next column. If VERTSPACE appears at the start of a column (PAGE,VS . . .), or after the column overflows, it does appear.

For an exact amount of vertical space, say 5 inches, that may be split over two columns, do the following:

```
<define &rem = remaining(2), –
comment distance remaining on page, –
if &rem > 5in, then, vertspace 5in, –
else, vertspace &rem, columnend, vertspace 5in–&rem, endif>
```

For an exact amount of vertical space that may not be broken across the columns and must appear at the top of the next page if it cannot fit on the current page, do the following:

```
<lineend, if remaining(2) > 5in, then, vertspace 5in, else>
```

<float top size 6.5in 4in comment>
<endif>

When a word containing a VERTSPACE overflows a column vertically, (perhaps being used to produce a subscript) then the word starts a new column. If it still doesn't fit, the VERTSPACE produced is only the amount of space available on the column. All subsequent VERTSPACE commands in the same word are ignored. Horizontal alignment is done on the last line of the column when the column overflows.

When a word containing a VERTSPACE overflows the line horizontally, the word is moved to a new line (or hyphenated if HYPHENATION is requested) before the VERTSPACE is done.

When successive VERTSPACE commands appear on a column, all the lengths requested appear. The VERTGAP, or VG command, however, produces only the largest length of all those requested. This command ends the current line, and produces a vertical space from the preceding baseline.

Text<VERTGAP 1IN>
<VERTGAP .5IN>Text2

produces:

Text

Text2

Notice that the baselines of 'Text' and 'Text2' are separated by a gap of 1 inch plus the LINESPACE used for the line containing 'Text2'.

If the VERTGAP requested is greater than the vertical space remaining on the column, the rest of the vertical gap does not appear at the top of the next column.

If a VERTGAP command is used at the very top or bottom of a column, or at the top of a reserve or float, it is ignored. Use VERTSPACE instead.

Spacing Text on a Column

The SPLIT command can be used to cause all the empty vertical space remaining on the column to appear when the command is used.

<**SP**LIT **VERT**ICAL>

This command causes no change to the horizontal position of the line.

<PAGE>
Text <LINEEND>

```
<SPLIT VERTICAL>
Text <LINEEND>
<PAGE>
```

puts the words ‘Text’ on the first and last line of the page.

The SPLIT VERTICAL command can be used several times per column. When this is done, the vertical white space on the page is divided by the number of times the SPLIT VERTICAL command appears.

```
<PAGE,SPLIT VERTICAL>
Text <SPLIT VERTICAL, PAGE>
```

puts the word ‘Text’ on the middle of the page. This command can be used only in **open text** — normal text which is placed on the page by TEXTFORM, as opposed to text which is placed in special locations, such as footnotes, keeps, floats or reserves. (These items are described later in the manual.)

Aligning Text Vertically in the Column

Whenever text overflows the column (or a one-column page), any extra vertical white space for the column appears at the bottom. More control over the vertical positioning of the text on the column is available by changing the variable VERTALIGNMENT, which is TOP by default. (But whenever SPLIT VERTICAL appears in a column or page, it is done instead of the vertical alignment for that page or column.) It can be:

TOP
BOTTOM
CENTRE or CENTER
BOTH

When VERTALIGNMENT=CENTRE, the text is centred vertically on the column when the column overflows or is ended with a COLUMNEND command. When VERTALIGNMENT=BOTTOM, the empty space on the column appears at the top of the column. Like the LINEEND command, alignment can also be specified on the COLUMNEND command if you want to change the alignment of a specific column:

```
<COLUMNEND CENTRE>
```

If you want to vertically align the column with the current value of VERTALIGNMENT, but you do not know what it is, use:

```
<COLUMNEND $VERTALIGNMENT>
```

where the ‘$’ causes TEXTFORM to insert the current value of VERTALIGNMENT into the command (see page 108 for more details about the ‘$’ operator). If the previous line will be part of a macro, use:

```
<'<COLUMNEND $VERTALIGNMENT>'>
```

If you want each column to be exactly the same length (vertically *justified*), this is a special case, because TEXTFORM assumes that there are no default points for extra

spaces to be added, even when VERTALIGNMENT=BOTH. (In the horizontal direction, blanks between words indicate where the space may be inserted.) You must indicate vertical justification points by inserting the VERTJUST command wherever you are willing to have extra space inserted; justification is not done unless these commands appear on the column. (No warning message is printed if justification is not done.)

The VERTJUST command, or VJ, is used to indicate where extra vertical space may be inserted when VERTALIGNMENT=BOTH. Only those columns which overflow or end with COLUMNEND BOTH are vertically justified. If COLUMNEND or PAGEEND ends a column, it is not vertically justified. (In the same manner, a line ended with LINEEND is not justified horizontally.) The last page of a document is not vertically justified.

Space is distributed evenly among all VJ commands in the column (presently, vertical justification is not done in reserves, floats, footnotes, or tables). Two VJ commands in a row produce twice as much space as one. The VERTJUST command should be given at the beginning or end of a formatted line, since it does not end the line.

Since there are no automatic vertical justification points, you will probably use macros to insert the VJ commands in the input. For instance, a major heading macro might include:

```
. . . LEND, VERTSPACE 3*LINESPACE, VJ, VJ, VJ, . . .
```

whereas a minor heading macro might include:

```
. . . LEND, VERTSPACE LINESPACE, VJ, . . .
```

so that more justification is inserted before major headings. You will probably want to insert VJ before and after block quotes and points. You may also insert VJ points at every paragraph with:

```
<DEFINE &NP='<LEND, VJ, NP>'>
```

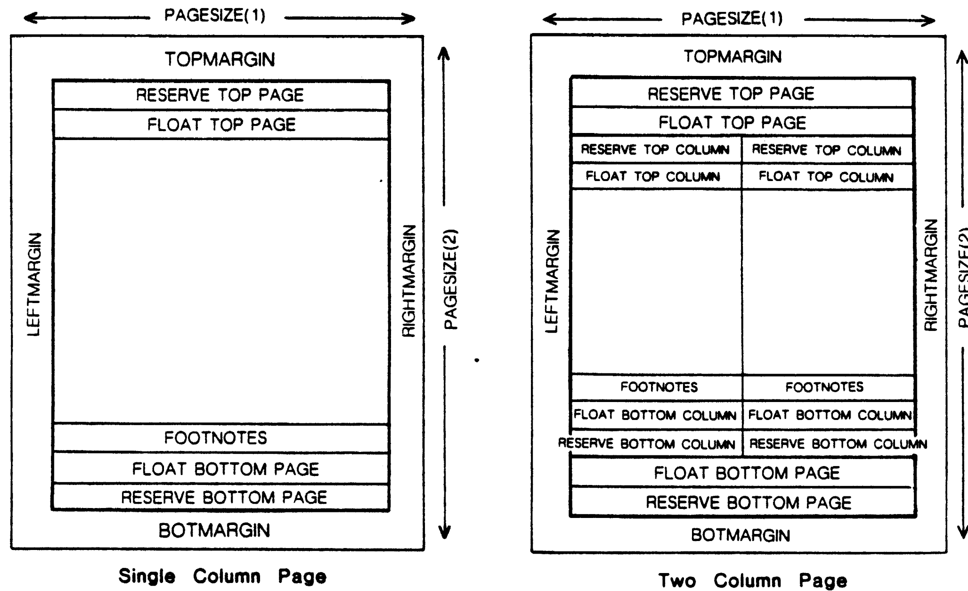
Note that VJ points are done even if they are the very first, or very last things in a column. If the &NP macro causes the column to overflow, there may be vertical justification at the bottom of the column. A more sophisticated &NP variable might do the following:

```
<lend, if remaining(2)>2*linespace, then, vj, eif, np>
```

Commands Which Include Input at Special Positions

In a file containing TEXTFORM commands, text and commands are formatted as they are encountered, and placed immediately on the **open text** area of the page. There are several commands that cause text and/or commands to be formatted at a later time in the document, or at a specific location on the page. These are discussed here, and include the FOOTNOTE, FLOAT, and RESERVE commands. AT and

DELAY commands also cause text to appear at special locations.



Default State

Before using these commands, you should be familiar with the concept of a **default state**. At the start of a run, there is a default linespace, underlining is off, font normal is in effect, alignment=left, etc. Your document may later change these values. It is necessary to ensure that all footnotes, floats and reserves (items that are not part of the open text area) have the same appearance, rather than some being single-spaced and others being double-spaced, for example. To do this, at the beginning of each, TEXTFORM saves the current values of all items in the default list, reverts to the default state, does the footnote, etc., and then restores the saved values. Within a footnote you might change any of these defaults, but the footnote always starts with the default values.

The items that are restored to their default value at the beginning of a footnote, float or reserve are:

wordspace is .1 inch
sentsep is .1 inch
typesize is .1389 inches, or 10 points
linespace is .1667 inches, or 12.0507 points
font is normal
characterset is odcharacterset
underline is off
underlinewordspace is false
underlinestring is '_'
indents are off
alignment is left

cap is false
keyboard is standard

When a top float or reserve appears, TEXTFORM positions down TSIZE to ensure that the text line on each page is in the same position, regardless of whether single or double spacing is used. At the bottom of a column or page, when a footnote, or bottom float or reserve begins, TEXTFORM is positioned down LINESPACE from the top boundary of the item. In either case, if you need to be positioned further down, use a VERTSPACE command. Then, you may also want to change alignment, typesize, or linespace to a new value. For example:

<FOOTNOTE,ALIGNMENT=BOTH>This is the footnote.
<ENDFOOTNOTE>

Footnotes at Bottom of the Column

The footnote commands let you enter the text of a footnote without knowing where you are in relation to the bottom of the column.

<**FOOT**NOTE> . . . <**E**ND**FOOT**NOTE>

Text entered between these commands is saved and printed at the bottom of the column, in the default state (see page 55) and is numbered with superscript arabic numerals.

The text of the footnote is entered immediately after the reference to it in the text—the FOOTNOTE command is used immediately after the text, without a blank space. However, to use the program most efficiently, there should be a blank space before the ENDFOOTNOTE command.

reference in the text.<foot>This is the text of footnote. <efoot> Text continues

produces in the text:

reference in the text.† Text continues

If a blank space appears between the FOOTNOTE command and the preceding word, or if the FOOTNOTE command is the first command on the line, this indicates where TEXTFORM may end the word. You may have justification inserted here, or worse, the footnote counter may appear on the following line. If a blank space appears between the FOOTNOTE command and the first word in the footnote, a word space appears between the reference in the footnote and the first word.

Footnotes cannot be used in tables, reserves, or bottom floats. If you forget an ENDFOOTNOTE command, and begin another footnote, you get the error *FLOAT or FOOTNOTE is already being built. You can't build one inside another, sorry.* There is a bug in the TEXTFORM program which causes it to occasionally miss the ENDFOOTNOTE command, even when it is entered correctly. The problem occurs when the last word of the footnote overflows the last line of the page, and the ENDFOOTNOTE command appears immediately after the word, without a

----
†This is the text of footnote.

preceding blank. If you get the errors *ENDFOOTNOTE command does not have a preceding FOOTNOTE command. Ignored.* or *End of file encountered while using a FOOTNOTE* you may be able to avoid the problem by inserting a blank before the ENDFOOTNOTE command.

Footnotes are separated from the rest of the text on the page with a separating line:

The commands to produce this line are in the variable FOOTSEP, which can be changed. Its default value is:

<FOOTSEP = '' <REP(18,'-'),LEND> ''>

On the X9700 output device, you can produce a solid line with:

<FOOTSEP = '' <REP(20, '_'), LEND> ''>

which repeats the character '_' 20 times. This produces 10 in the output, since it is a meta-character. If the character set has a LIGHTRL character, use it:

<FOOTSEP = '' <REP(18, LIGHTRL),LEND> ''>

If FOOTSEP is changed, the output it produces must not exceed one line. If it does, this may cause the error *RESERVEs on page leave no room for text.*

Footnotes on a column can be separated by a string or space if you change FOOTDIVIDE. Although it is initially '', you can place one blank line between each footnote with:

<FOOTDIVIDE = '<vertspace linespace>'>

To separate each footnote with '***':

<FOOTDIVIDE = '***<lend>'>

Like FOOTSEP, the output from FOOTDIVIDE should not be more than one line high.

When TEXTFORM cannot fit the entire footnote on the column, the word "cont'd" and the rest of the footnote appear at the bottom of the next column. The word "cont'd" is in the variable FOOTCONTINUE, and can be changed. For example:

<FOOTCONTINUE = 'continued . . . ' >

Each footnote is numbered consecutively throughout the document. The command <AT ENDOFPAGE RESETFOOTNOTE> or <AT EOP RSFOOT> causes footnotes to be numbered from [1] on each page. The AT command is described in more detail on page 69.

You can make further changes to how the footnotes are numbered, by adjusting the FOOTCTR and FOOTINDEX variables. By default, FOOTCTR = 1. It is incremented and appears as a superscript for each footnote.

If FOOTCTR is changed to a structure, it is indexed by FOOTINDEX, and the items in the structure are then used to number the footnotes. For example, footnotes are frequently itemized with the characters * and †. If there are more than two footnotes in the column, multiples of these characters are used: **, ††, ***, and so on. FOOTINDEX indexes the structure element which appears with the footnote. When FOOTINDEX exceeds the number of elements in the structure, multiples of the structure elements are used.

```
<AT ENDOFPAGE RESETFOOTNOTE>
<FOOTINDEX = 1>
<FOOTCTR = ( '*', '<DAGGER>' )>

This<foot>FOOTINDEX is 1 for first footnote on the page <efoot> is
an example<foot>FOOTINDEX is now 2 <efoot> of how
FOOTCTR<foot>FOOTINDEX is 3 so FOOTCTR(1) will appear twice
<efoot> FOOTCTR can be changed.
```

produces:

> This* is an example† of how** FOOTCTR can be changed.

If you do not want to have footnotes numbered, enter the command:

```
<FOOTCTR = (' ')>
```

Footnotes at the End of a Chapter

The FOOTNOTE command prints footnotes at the bottom of each column. You may prefer to collect them until the end of each chapter, or perhaps until the end of the document. Instructions for doing this follow. Instead of using FOOTNOTE and ENDFOOTNOTE, enter each footnote as follows:

```
<&FOOT>This is a footnote<$&EFOOT>
```

When you want the footnotes to be printed, enter the command:

```
<&PRINTNOTES>
```

Here are the instructions to use (they are also available online; see TXFM:INDEX for the file name):

```
<DISABLE MESSAGE 227 228>
<LOAD TOSUP &PRINTSUP –
     TAKES STYPE LEN STRING –
     RETURNS R0 PTR VALUE LEN STRING>
<COMMENT Ignore the warning from this command>

<DEFINE MACRO &PRINTNOTES, –
     LINEEND, –
```

----
*FOOTINDEX is 1 for first footnote on the page
†FOOTINDEX is now 2
**FOOTINDEX is 3 so FOOTCTR(1) will appear twice

```
     FOR '&FN=1' UNTIL '&FNMAX' DO &PRINTFOOT, –
     &FN = 1, –
     EDEF MACRO &PRINTNOTES>

<DEFINE &FN = 1, DEFINE &FNMAX>
<DEFINE &PRINTFOOT = '<LINEEND,VS LINESPACE, $&AP,
ERASE $&AP>' >

<DEFINE MACRO &FOOT, –
     DEFINE &EFOOT = &XEFOOT, –
     &PRINTSUP(&FN), –
     DEFINE MACRO $&AP, &FN, HORSPACE WORDSPACE, –
     EDEF MACRO &FOOT>

<DEFINE &XEFOOT = '$&EAP, &FNMAX=&FN, &FN=&FN+1,
ERASE &EFOOT'>
<DEFINE &AP = ('&',@&FN)>
<DEFINE &EAP = ( 'EDEF MACRO &', @&FN )>
```

The HORSPACE WORDSPACE command in the above example can be replaced by '. ' if you require punctuation in the list of footnotes after the numbers.

Here is an example using the above intructions. Be especially careful to end each footnote you enter with <$&EFOOT> because errors in using these instructions can be costly.

```
Chapter 1
<LINEEND CENTRE> This is the start of a chapter that contains
some footnotes.<&FOOT> This is a footnote<$&EFOOT>
This is some text that will appear in the chapter.<&FOOT> And
another footnote.<$&EFOOT>
Now more text on the page, and then print the footnotes.
<&PRINTNOTES>
<LEND,VS 2*LINESPACE> Chapter 2
<LINEEND CENTRE> This is another chapter that will contain more
footnotes<&FOOT> This footnote will begin at one.<$&EFOOT> and
this chapter continues. Now, here are the footnotes for this chapter.
<&PRINTNOTES>
```

produces:

Chapter 1

This is the start of a chapter that contains footnotes.[1] This is some text that will appear in the chapter.[2] Now more text on the page, and then print the footnotes.

[1] This is a footnote

[2] And another footnote.

Chapter 2

This is another chapter that will contain more footnotes[1] and this chapter continues. Now, here are the footnotes for this chapter.

[1] This footnote will begin at one.

Input at a Specific Location

The FLOAT command can be used to have a portion of text, or blank space for an insertion, appear in a special position, while the regular input text continues to fill the current column. The text after the FLOAT command may appear before, around, or after the floated text appears in the output, depending on how the float is specified. In contrast, the KEEP command ensures only that portions of text appear in the same column. If the text must begin in a new column to do this, the bottom of the previous column is left blank.

Floats save text until a specified condition becomes true. If some text is in a variable, and you want the text at the top of a page, use the following FLOAT command:

```
<DEF &TOP = 'A float from previous page. <LEND R, SP ". ",
LEND, VS 4MM>'>
<FLOAT &TOP>
```

The contents of &TOP are not evaluated until the FLOAT appears on the page. This means that the current page is finished, the next page is started, and then TEXTFORM checks for top floats. If any are waiting, such as &TOP, the *contents* of &TOP are then included as TEXTFORM input. If &TOP is changed after the float command, but before the page ends, the *new* contents of &TOP are evaluated on the next page.

All float text appears in the default state, as described on page 55. The first floats specified appear closest to the margins. If several FLOAT TOP commands are given, the first one appears at the top of the next page (after the reserves, if any) followed immediately by the second, and so on. TEXTFORM does not check that the entire FLOAT will fit on the same page unless specific keywords are used in the command. The keywords of the FLOAT command let you control where the text is placed on the page, and the size of the space in which the text appears (this is useful if you want to float blank space instead of text). The complete list of keywords are in Appendix 1.

Examples of Floats

To leave a space of fifteen blank lines at the bottom of a page which is 6.5 inches wide, the commands are:

```
<DEFINE &FL>
<FLOAT BOTTOM PAGE SIZE 6.5IN 15*LINESPACE &FL>
```

If the horizontal width of the page is unknown, it may be given as:

```
<FL B S DEFAULT 15*LINESPACE &FL>
```

To float an entire blank page (this is not the same as including the commands

A float from previous page.

. . . .

PAGE, PAGEEND in the input, which may leave part of the current page empty) use the following commands. It is not necessary to do this exactly at the end of a page. It can be done anywhere, and several times in a row if you need more than one blank page.

```
<FLOAT VERY TOP PAGE PAGEEND>
```

The size of a bottom float must be specified (if you need to estimate, estimate on the high side). Bottom floats do not require PAGEEND commands (it causes an error) because no text appears on the page after the float (except footnotes).

```
<DEFINE MACRO &BTM>
    This text appears at the bottom of a column.
<ENDDEFINE MACRO &BTM>
<FLOAT BOTTOM COLUMN SIZE 3.5IN .33IN &BTM>
```

A convenient method of handling figures and tables is to put each one in a macro with a suitable name, and then to float the macro when the figure is referenced. For example, if you have a table called &TAB, you might do the following:

```
<DEFINE MACRO &FIG1>
  <DEFINE TABLE &TAB>
  <COLUMNS = 3>
  <ENDDEFINE TABLE &TAB>
   <USE &TAB>
   This <TAB 2> is a sample <TAB 3>table.
   <USE, &TAB>
<ENDDEFINE MACRO &FIG1>
```

In the text, when you refer to Figure 1, you can then float the macro you have defined:

```
. . .as illustrated in Figure 1. <FLOAT TOP &FIG1>
```

You can build macros which float space to the top of the next page when there is not enough room to leave the space on the current page. This is useful if you are leaving space to insert photographs or figures. The following macro, &FIG, takes a parameter which specifies the vertical dimension of the space to be left blank. The macro calculates the page width between margins, so it can be used even when you do not know the pagesize. If there is enough room on the current page, a blank space is left. If there is not enough room on the page, the message ‘(see following page)’ is printed, and the space is floated to the top of the next page.

```
<define &message = '(see following page)'>    <define &figctr = 1>
<attribute &figctr incr = '1'>
<define &figtitle = '<vs linespace>figure <&figctr, lend centre>'>
<define macro &fig, –
  if remaining(2) >= par(1) then, –
  &figtitle, vertspace par(1)–(3*linespace), –
  else, &message, –
  float size default par(1) &figtitle, –
  endif, –
enddefine macro &fig>
some text . . .
<&fig(1inch)>The text resumes one inch from the preceding text on
the page.
```

produces:

some text . . .

Figure 1

The text resumes one inch from the preceding text on the page.

In the next example, &FLFIG generates a float command of a specified size when given three parameters: vertical depth of float (if omitted, 18*linespace+.5IN), number for figure, and the title to appear at the bottom of the figure.

```
<disable message 228>
<define &flctr=1>

<define macro &flfig, –
if par(1)='',then, –
     local &distance=18*linespace+.5in, –
     def $&fl='<vs 18*linespace>':par(2):par(3):'<lend c>', –
else, –
     local &distance=par(1)+.5in, –
     def $&fl='<vs ':par(1):'>':par(2):par(3):'<lend c>', –
endif, –
if remaining(2) < &distance, –
     then fl top page size default &distance $&fl, –
     else fl bottom page size default &distance $&fl, –
endif, –
&flctr=&flctr+1, –
enddefine macro &flfig>

<define &fl=('&flfigno',@&flctr)>
```

Use it as follows:

```
<&FLFIG(10*linespace, '1', 'Title of Figure')>
<&FLFIG(15*linespace, '2', 'Title of Next Figure')>
```

Include Float

The INCLUDE command includes items which have not yet appeared, such as FLOATs. This is useful at the end of a chapter or section.

<**INC**LUDE **FL**OAT [VERY] [kwd2] [kwd3] [kwd4]>

It takes up to five keywords, where the first is FLOAT and the others are the same as for the FLOAT command. Only the FLOAT keyword is compulsory; all the other keywords are optional. If a keyword is not given, all floats waiting are included:

<INCLUDE FLOAT>

If neither TOP nor BOTTOM is specified, then floats for both the TOP and the BOTTOM are included. The same is true for the FRONT or BACK pair of keywords as well as PAGE or COLUMN.

INCLUDE FLOAT ends the current line, and starts a column, if one has not been started already. Then, if a float is found, it places it on the column and ends the column, unless it is the last column containing a top float. While processing the requested floats, the INCLUDE FLOAT command may cause other floats to appear, along with the ones requested. For example, the command INCLUDE FLOAT TOP does not prevent bottom floats from appearing if any are pending and will fit.

At the end of a run, all floats that have not yet appeared are automatically included. However, if a float has been given with dimensions that are too large to fit on a page, it will not appear.

Information About Floats

FLOAT_INFO provides information about pending floats. It is similar to RESERVE_INFO, described on page 67.

Repeated Input at a Specific Location

The FLOAT command prints text in a special position on the page. This floated text appears one time only. If text is to appear in a special position on *every* page, use the RESERVE command instead (see Appendix 1 for the command syntax, which is like the FLOAT command). This is how page numbers, headers, and footers are produced.

Examples of Reserves

In the following example, space for a title is reserved at the top of the page. First, a variable is defined to contain the title.

<DEFINE &TITLE = 'Rules of Court<LINEEND CENTRE,VS LINESPACE>'>

To cause this variable to be printed at the top of every page, the command is:

<RESERVE TOP &TITLE>

This reserve uses two lines at the top of every page. One line contains the title ‘Rules

of Court'; one empty line follows as a result of the VS command. Reserves appear on the page after the top margin, or before the bottom margin. If TOPMARGIN is one inch (six lines), text begins on the ninth line of the page in this example. Those reserves given first appear closest to the margins. As with floats, the reserve starts in the default state.

Margins are often reduced if a reserve is being used. In the above example, the TOPMARGIN could be reduced to .66 inch to compensate for the .33 inch occupied by &TITLE.

The value of THISPAGE can be used to align the page number:

```
<DEFINE &TOP = '<PNCTR, LINEEND $THISPAGE>'>
<TOPMARGIN = .5IN>
<RESERVE TOP SIZE 6.5IN .5IN &TOP>
```

produces either a LINEEND LEFT or a LINEEND RIGHT command depending on the *current* value of THISPAGE.

Notice that the top margin has been reduced by the vertical size of the reserve. As a result, the total white space at the top of the page will be 1 inch, although the page numbers appears in this space.

The following example prints the page number in the right corner of the page, but also puts a centred title on the same line:

```
<DEFINE MACRO &HEADER, –
   PNCTR, LINEEND RIGHT, VERTSPACE –LINESPACE, –
   'Title Goes Here', LINEEND CENTRE, –
ENDDEFINE MACRO &HEADER>
<RESERVE TOP &HEADER>
```

This would produce:

```
                    Title Goes Here                    23
```

It is easiest to change the reserve if it is produced by a variable rather than a macro. For example, set up a reserve which prints the variable &TITLE. The following macro ends the current page, changes &TITLE so that it prints nothing, prints a chapter heading at the top of a new page, and then resets &TITLE to appear on subsequent pages:

```
<DEFINE MACRO &CHAP, –
   PAGEEND, &TITLE=' ', PAR(1), LEND, VS LINESPACE, –
   &TITLE = PAR(1), –
ENDDEFINE MACRO &CHAP>
```

When you print the page number at the bottom of the page, remember to include vertical space before PNCTR is printed, so that there is blank space between the bottom line of text on the page and the page number.

```
<DEFINE &BOTCEN = '<VERTSPACE LINESPACE, PNCTR, LEND
C>'>
```

<RESERVE BOTTOM SIZE 6.5IN .5IN &BOTCEN>

If you want the page number printed as a roman numeral, use the ATTRIBUTE command, on page 123.

<ATT PNCTR DISPLAY ROMAN LC>

The contents of the reserved item can be changed at any time if contained in a variable. In a bottom reserve, this change is reflected at the bottom of the page.

Because a top reserve is printed whenever a PAGE command causes a new page to begin, or when a page overflows, the change does not appear until the next page. However, a TOP reserve can be changed after a PAGEEND command and takes effect for the page following. You can issue negative vertical space commands from bottom reserves to print text in a top reserve. This is illustrated in the dictionary-type example which follows.

To change the contents of the reserve before the next page begins, you might use the commands:

<PAGEEND, &TITLE = 'New Section<LINEEND CENTRE,VS LINESPACE>'>

The following commands will not work, because the PAGE command will make the top reserve appear before the contents of &TITLE are changed.

<PAGE, &TITLE = 'New Section<LEND C, VS LINESPACE>'>

If &TITLE is changed to '' the reserve does not print any text, but it is still in effect. It is not as easy to change the reserve if it is a macro, because the macro must be erased and defined again. (The command is ERASE name, as described on page 116.) To stop a reserve entirely, it is 'suspended', described next.

The full form of the RESERVE command, shown in Appendix 1, is similar to the FLOAT command. There are several differences: by default, reserves appear on the page instead of the column; reserves cannot use the VERY keyword; and reserves must have a name so that they can be suspended.

Stopping Reserves

The SUSPEND command stops reserves:

<**SUS**PEND kwd>

kwd
indicates which facility to suspend. In the case of RESERVES, kwd is

RESERVE [kwd1] [name1]

kwd1
may be provided to specify a 'name1' which appears on more than one reserve list. If 'kwd1' is omitted, 'name1' is removed from all reserve lists.

TOP – remove from the RESERVE TOP list
BOTTOM – remove from the RESERVE BOTTOM list
FRONT – remove from the RESERVE FRONT list
BACK – remove from the RESERVE BACK list

name1
if specified, indicates the name to remove from from the reserve list. If ‘name1’ is not found, it generates the error *No reserve by that name. SUSPEND command is ignored.* If ‘name1’ is not specified, all names on the list are removed.

To stop all reserves in effect at the top of the page:

<SUSPEND RESERVE TOP>

To remove the name &TOP from all reserve lists:

<SUSPEND RESERVE &TOP>

To remove the name &TOP from all reserve TOP lists:

<SUSPEND RESERVE TOP &TOP>

Example of Dictionary-Type Page Titles

In a dictionary-type document, you may want the first and last keyword on the page to appear at the top of the page. However, the top reserve does not know what the last word on the page will be. To solve this problem, use a bottom reserve which spaces vertically back to the top of the page.

If your pagesize is (8.5IN,11IN), set top and bottom margins to zero. Reserve a 1 inch space at the top and bottom of the page. When the bottom reserve begins, it is ready to begin printing on the *first* line of the reserve. At this point, a VERTSPACE –10IN command moves the baseline (the imaginary line on which the letters seem to rest) to the first line of the top reserve, i.e. the top line of the page. A VERTSPACE –9.5IN command moves the baseline to the fourth line of the top reserve.

Be sure that the negative VERTSPACE command positions you in a blank part of the page (i.e. in a top reserve). If the negative VERTSPACE requested would take you past the top of the page, the following error appears: *You cannot back up any farther than the top of the page. Stopped at the top of the page.*

To print a dictionary-type heading at the top of each page, two variables are used.

- One remembers only the *first* entry on the page. In the following example, &SETFIRST sets &FIRST once per page, then it sets itself to NULL. The bottom reserve restores &SETFIRST so that it will reset &FIRST at the top of the next page.
- A second variable must be set with each successive entry so that it contains the last entry on the page whenever the page overflows. In the following example, the entry macro &E sets &LAST each time it prints a new entry.

```
<pagesize = (8.5in,11in)>
<topmargin = 0in>
<botmargin = 0in>
<define &top>
<reserve top     page size 6.5in 1in &top>
<reserve bottom page size 6.5in 1in &bot>
<define macro &bot, –
   vertspace –9.5in, &first, '–', &last, –
   &setfirst = '<&first = par(1), &setfirst = " " >', –
enddefine macro &bot>
<define &first, define &last>
<define &setfirst = '<&first = par(1), &setfirst = " " > '>
<define macro &e, –
   line, vertspace linespace, font 2, par(1), font 1, –
   &setfirst, &last = par(1), –
enddefine macro &e>
```

After the above commands, each entry would appear as:

```
<&E('cats')> . . .
<&E('dogs')> . . .
.
.
<&E('rats')> . . .
<&E('rhinos')> . . .
```

If ‘dogs’ and ‘rhinos’ were the first and last entries on the page, ‘dogs–rhinos’ would appear in the top reserve area.

Information About Reserves

RESERVE_INFO provides information about reserves presently in effect. It is used with a list of search terms which it uses to limit the results provided. The results are in the form of a structure of strings, with each separate string containing the complete reserve command which was found. The SEPARATE function can be used to extract single words from these strings. FLOAT_INFO provides similar information; the differences are described below.

The list of search terms can include:

TOP
BOTTOM
FRONT
BACK
PAGE
COLUMN
SIZE
name of reserve

For these examples, there are these reserves on the page.

```
RESERVE TOP PAGE &1
RESERVE TOP FRONT PAGE &2
RESERVE TOP BACK PAGE &3
```

```
RESERVE  BOTTOM  PAGE  SIZE  6.5IN  1IN  &4

<&RESULT  =  RESERVE_INFO>
```

No parameter was given, so &RESULT is now a four element stucture containing

```
1      RESERVE  TOP  PAGE  &1
2      RESERVE  TOP  FRONT  PAGE  &2
3      RESERVE  TOP  BACK  PAGE  &3
4      RESERVE  BOTTOM  PAGE  SIZE  6.5IN  1IN  &4
```

Now, only TOP reserves are requested, so the result of

```
<&RESULT  =  RESERVE_INFO("TOP")>
```

&RESULT is a three element structure containing

```
1      RESERVE  TOP  PAGE  &1
2      RESERVE  TOP  FRONT  PAGE  &2
3      RESERVE  TOP  BACK  PAGE  &3
```

The next test uses two parameters. They ask information about all the reserves that are both TOP and FRONT. Note that &RESULT contains information on all reserves which are done for front pages, not just those reserves which are done for only front pages.

```
<&RESULT  =  RESERVE_INFO("TOP","FRONT")>
```

produces:

```
1      RESERVE  TOP  PAGE  &1
2      RESERVE  TOP  FRONT  PAGE  &2
```

In the following example, all column reserves will be returned, but since there aren't any defined in this example, &RESULT contains a null string.

```
<&RESULT  =  RESERVE_INFO("COLUMN")>
```

Now, any reserve with the name &3 is found. There may be several with that name, but in these examples there are only one, so

```
<&RESULT  =  RESERVE_INFO("&3")>
```

produces:

```
3      RESERVE  TOP  BACK  PAGE  &3
```

Note:

1. If nothing matches the search list, the result is not a structure with no elements, but is a null string instead.
2. The search terms are not checked for a rigorous syntax like they are on the FLOAT or RESERVE command. If you mistype one of the terms, these functions will assume that you meant that term to be a name to look for. Also, the search

terms do not have to be in the same order as they are on the RESERVE or FLOAT command.

3. The command returned is not identical to the one you typed in. If you allowed some of the keyword to use their default values, those will be included. For example, if you said FLOAT &name it would return
   FLOAT TOP COLUMN &name.
4. In unusual circumstances, it is possible to have so many RESERVEs or FLOATs queued that the resulting structure becomes too large and overflows, causing TEXTFORM to terminate.
5. FLOATs may be found up until the time they are used. For instance, if you are in the middle of a column, the FLOATs for the TOP have already been used and removed. However, the FLOATs for the BOTTOM of the column have not yet been done, so they can be found by FLOAT_INFO.

AT Points

If you want to perform a specific action repeatedly after an event occurs, use the AT command. The event can be:

an assignment to a variable (e.g. <&A=1>)
when a name is referenced (e.g. <PEND>)
a specific text character
end of word
end of line
start or end of column
end of page
end of file

The first three cases are done when TEXTFORM encounters the specific item in your input file, and are called **synchronous** ats. The others are done when a condition becomes true during the TEXTFORM run—at the end of a line or end of a page. These are called **asynchronous** ats, and occur whether or not you use a specific command to force the condition to become true. The format of the command is:

<AT kwd name>

kwd

ASSIGN name2
END**O**F**C**OLUMN
END**O**F**F**ILE
END**O**F**L**INE
END**O**F**P**AGE
END**O**F**W**ORD
REFERENCE name2
START**O**F**C**OLUMN
TEXT**C**HARACTER c

name
is done, even within tables, floats, and reserves. TEXTFORM does not return to the default state before including the name. As many ATs may be queued as desired. However, do not use ATs when INPUTMODE is ASIS. More than one AT may be queued for each 'kwd'; however they must be queued in separate commands. The ATs

queued for a particular 'kwd' will be carried out in a last-queued first-done order. The 'name' in an AT will automatically be de-queued if for some reason the 'name' cannot be queued. This causes the error *&name is not defined. Deleted from AT-list.*

When an AT ENDOF. . . generates text, the text appears in the next open text area. For example, AT ENDOFPAGE text appears at the first open text area of the next page, after reserves and floats.

Examples of ATs

Footnote counters are reset by using an AT. By default, FOOTCTR is incremented throughout the document. To reset the counter to 1 at the end of every page, enter the following command at the beginning of the document:

```
<AT ENDOFPAGE RESETFOOTNOTE>
```

WORDSPACE and SENTSEP are often adjusted when TYPESIZE or ALIGNMENT are changed. The following example shows how to do this automatically by using AT ASSIGN. AT ASSIGN ‘name2’ occurs whenever a value is assigned to ‘name2’. It is only meaningful when ‘name2’ is a variable name. This condition does not occur when TEXTFORM changes a system variable.

```
<def &a=tsize>
<def &b='<wordspace=wordspace*(tsize/&a),&a=tsize>'>
<at reference typesize &b>
<at reference typ &b>

<def ma &c, –
   if alignment=both then wordspace=textwidth('i'), –
   else wordspace=textwidth('n'), eif, –
edef ma &c>
<at assign alignment &c>
```

AT ENDOFFILE name comes true when TEXTFORM reads an end-of-file from SCARDS, i.e. finishes reading your file. The following commands cause the index to be automatically printed at the end of the run.

```
<DEFINE &PRINTINDEX = '<X(0)>'>
<AT ENDOFFILE &PRINTINDEX>
```

AT ENDOFLINE name occurs whenever the output overflows from one line to the next, or when a line is ended as the result of a command. To prevent formatting errors, TEXTFORM does not do AT ENDOFLINEs in bottom reserves. The following example automatically numbers lines.

```
<DEF &LINES = 1,ATTRIBUTE &LINES INCR = '1'>
<DEF &LNCT = '<&LINES>'>
<AT ENDOFLINE &LNCT>
<NEWPARA>Whenever a line overflows to the next line, the contents
of &LNCT are inserted in the input. If &LNCT causes text to be
printed, this text will appear at the beginning of the next line.
<SUSPEND AT ENDOFLINE &LNCT>
```

produces:

1
2

Whenever a line overflows to the next line, the contents of
3 &LNCT are inserted in the input. If &LNCT causes text to be
4 printed, this text will appear at the beginning of the next line.

AT ENDOFWORD 'name' is done at the end of every word. It can be used to count the number of words in a body of text. AT REFERENCE name2 occurs whenever 'name2' is referenced. But AT REFERENCE NP is not done when NEWPARA is referenced.

<DEFINE &COUNT = 1>
<DEFINE &PARAS='<&COUNT,''. '',&COUNT=&COUNT+1>'>
<AT REFERENCE NEWPARA &PARAS>
<NEWPARA> Whenever the command NEWPARA appears in the text, the contents of &PARAS are inserted in the text.
<NEWPARA>This causes the number to appear before the text of each paragraph.
<LINEEND, &PARAS= ' ', SUSPEND AT REFERENCE NEWPARA &PARAS>
<NEWPARA>However, suspending the at-point requires the command NEWPARA to be used. This causes the at-point condition to become true again.

produces:

1. Whenever the command NEWPARA appears in the text, the contents of &PARAS are inserted in the text.

2. This causes the number to appear before the text of each paragraph.

However, suspending the at-point requires the command NEWPARA to be used. This causes the at-point condition to become true again.

AT STARTOFCOLUMN name becomes true when TEXTFORM begins a column. See page 152 for details. AT TEXTCHARACTER c has been discussed earlier on page 42.

Stopping ATs

The SUSPEND command stops ATs:

<**SUS**PEND kwd>

kwd
indicates which facility to suspend. In the case of ATs, kwd is

AT kwd2 [name2]

kwd2
indicates which kind of AT to suspend (e.g., ENDOFPAGE), and must be included.

name2
if given, causes TEXTFORM to remove only that name from the AT list. If not given, all names are removed.

Information About ATs

AT_INFO provides information about which AT points are active. It is used with the same keywords as the AT command. For example:

```
<define &dis = '<display thispage>', –
  at eop &dis, –
  at endofpage resetfootnote>

<define &show = at_info('eop')>
<display &show>
```

1 RESETFOOTNOTE
2 &DIS

Delayed Input

The DELAY command allows an action to be delayed for some measure. The delay measure may be a given distance or a number of lines. An action can be delayed vertically or horizontally. The format of the command is:

```
<DELAY [kwd] kwd2=expression name>
```

kwd
is the direction to delay.

> **V**ERTICAL – vertically down from the current position.
> **H**ORIZONTAL – horizontally to the right of the current position.

kwd2
is the distance to delay.

> **LEN**GTH – if used, ‘expression’ is the distance to wait before doing the delay.
> LINES – if used, ‘expression’ is the number of lines of text to wait before doing the delay. This can only be used with vertical delay.

name
is the name of the item (variable or macro) containing the text and/or commands of the delay. If it is a macro, parameters are not allowed. If it is a variable, the variable can be changed or assigned a null value.

In the horizontal direction, the counting of the delay measure is done at the end of every word and after all HORSPACE commands. In the vertical direction, the counting of the delay measure is done at the start of a line and also after all VERTSPACE, VERTGAP, and SPLIT VERTICAL commands. A delay will awake when at some point (a line boundary or word boundary), the vertical or horizontal distance is exceeded. However, this may not be the exact distance specified in the DELAY command.

Page Appearance

<DEFINE &ACTION='<F 3>Delay Stuff. <F>'>
This is some text <L, DELAY LINES=2 &ACTION>
This is some more text <L>
This is text which appears just before the delay stuff. <L>
And this is text which appears after the delay stuff.

produces:

This is some text
This is some more text
This is text which appears just before the delay stuff.
Delay Stuff. And this is text which appears after the delay stuff.

If two DELAYs come true at the same location, for example:

<L,DELAY LINES=2 &ONE>text
<L,DELAY LINES=1 &TWO>text

the first DELAY specified appears first. However, if two DELAYs come true at the same location, but one has a more negative value, for example:

<DELAY HORIZONTAL LENGTH = –1IN &1>
<DELAY HORIZONTAL LENGTH = –2IN &2>

the one with the more negative value appears first. If a DELAY is given and an end of file is reached before the DELAY comes true, then the DELAY will not appear and TEXTFORM will issue the warning *DELAY was pending but was not done.*

DELAYs are active only in the type of item in which they are given. For example, if a DELAY is given in normal text, it is temporarily suspended whenever footnotes, floats and reserves are being done. TEXTFORM will continue to check for the DELAY to come true when it resumes formatting normal text. However, if DELAYs are given within a footnote, float or reserve and if they do not come true before that item is done, the pending DELAYs would be cancelled.

PRODUCING COLUMNS ON THE PAGE

Columns on the page can be produced in several ways—tables, logical pages, and input mode. These three techniques are described in this section.

If you want TEXTFORM to fill a column to the bottom of the page, and then begin at the top of the next column, use logical pages. However, if you want control over the vertical position of the text in each column, you should treat the text as a table. When you want to enter text exactly as it is to be produced, change the INPUTMODE variable to tell TEXTFORM to stop formatting the text.

Before using these options, you should know several new TEXTFORM terms. The **physical page** is the size of the page TEXTFORM works with, although the exact sheet may be larger (as in the 1403 output device). One physical page may be divided into several **logical pages**, or columns. On a physical page or logical page, tables may produce further columns. The command which tells TEXTFORM where to place text — on the physical page, the logical page, or in the columns of a table, is the USE command. It has the format:

<USE [name]>

name

- if not given, or if PHYSICALPAGE, causes TEXTFORM to place text on the physical page. Whenever the physical page becomes full, or when a PAGEEND command is encountered, TEXTFORM sends the page to the output device.
- if the name of a logical page, as in USE &2COL, causes TEXTFORM to place text according to the dimensions of logical page called &2COL. Whenever that page becomes full, or is ended with a PAGEEND command, the page is sent to the output device.
- if the name of a table, causes TEXTFORM to direct text to column 1 of that table.

Whenever the USE name command is given, TEXTFORM changes CURLP to an upper case string containing ‘name’. If ‘name’ is not given, CURLP contains PHYSICALPAGE. So CURLP can contain one of:

- the name of the current logical page
- the name of the current table
- PHYSICALPAGE

Logical Pages

The following example defines and uses a logical page.

> First, define the logical page. The one in this example has two columns, with .5 inch space between columns.
>
> <DEFINE LOGICALPAGE &TWOCOL, –
>   COLUMNS = 2, –
>   DEFCOLGAP = .5IN, –
> ENDDEFINE LOGICALPAGE &TWOCOL>
>
> To format text in two columns, USE the logical page:
> <PAGEEND>
>
> <USE &TWOCOL>
>
> This text is formatted in two columns. The USE command positions at the beginning of the first column (unless the logical page already has text on it). The command COLUMN
>
> <COLUMN>
>
> causes TEXTFORM to begin at the top of the next column. If the COLUMN command appears in the last column of the page, it has the same effect as a PAGE command. Now end the two column page with PAGEEND and continue on the physical page.
> <PAGEEND, USE PHYSICALPAGE>

produces:

> First, define the logical page. The one in this example has two columns, with .5 inch space between columns. To format text in two columns, USE the logical page:
>
> This text is formatted in two columns. The USE command positions at the beginning of the first column (unless the logical page already has text on it). The command COLUMN
>
> causes TEXTFORM to begin at the top of the next column. If the COLUMN command appears in the last column of the page, it has the same effect as a PAGE command. Now end the two column page with PAGEEND and continue on the physical page.

Defining the Logical Page

Text can be formatted in one or more columns on the physical page by defining and using a logical page. To produce text in columns 1) define the appearance of the logical page by specifying the number of columns and where they appear on the page 2) tell TEXTFORM to place text on the logical page with the USE command 3) use the COLUMN and COLUMNEND commands to control your position on the column

The logical page is built within the total area available for text on the physical page, i.e., within the margins. Any number of logical pages can be defined; each logical page can have one or more columns. If a reserve, float, or footnote is in effect on the physical page, it has no effect on the dimensions of the logical page being defined. If the reserves are producing page numbers on the physical page, they will not appear on the logical page.

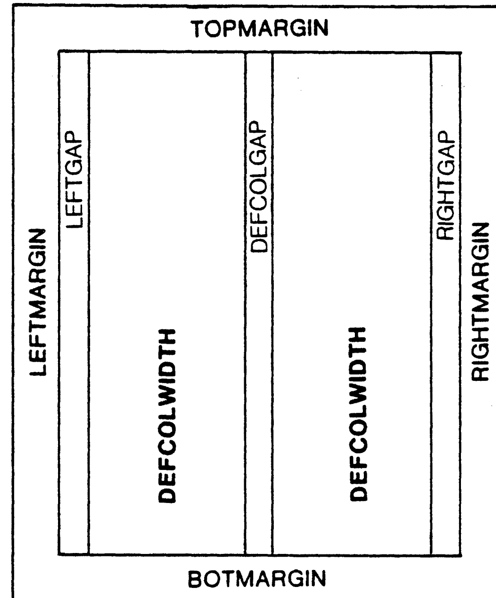
The following list describes keywords which are valid within the logical page definition between the commands:

<**DEF**INE [kwd] **L**OGICAL**P**AGE name>
keywords ...
<**E**ND**DEF**INE LOGICAL**P**AGE name>

kwd
may be one of the AXR or NXR. See cross references on page 176.

name
is the name of the logical page to be defined. Any number of logical pages can be defined.

keywords
Information within parentheses is the default used if the item is not specified. Any unspecified column width or gap size information is calculated when the definition is closed with the ENDDEFINE LOGICALPAGE command.



COLUMNS = expression (= 1)
The logical page will have ‘expression’ columns. To further describe each column, the DEFINE COLUMN command (see page 91) may be used within the logical page definition. If COLUMNS= is not specified, the number of columns is taken from the highest column defined. COLUMNS= can appear before or after DEFINE COLUMN. If DEFINE COLUMN is not used, the width of each column is taken from DEFCOLWIDTH, and the width of the column gaps from DEFCOLGAP.

If the specified number of columns cannot fit across the page, an error is issued: *Defined columns and their gaps leave no room for text. Automatic sizing invoked.* Then the column width is adjusted to fit the desired number of columns across the page.

DEFINE COLUMN . . .
See the DEFINE COLUMN command on page 91.

DEFCOLWIDTH = length (depends on page width)
This is the column width. If it is not specified, TEXTFORM calculates the column width, fitting the columns evenly. If neither DEFCOLGAP and DEFCOLWIDTH are specified, DEFCOLGAP is set to 10% of the column width necessary to use the entire page width.

DEFCOLGAP = length (depends on page width)
This is the gap between columns. If DEFCOLGAP is not specified (or is set to zero or a negative number), TEXTFORM calculates the column gap, fitting the columns evenly across the page.

LEFTGAP = length (= 0)
This is the gap between the left margin and column 1. If a negative number is specified, TEXTFORM makes the gap 10% of the column width.

RIGHTGAP = length (= 0)
This is the gap between the last column and the right margin. If a negative number is specified, TEXTFORM makes the gap 10% of the column width.

COMMENT body
A comment may appear.

PAGESIZE = length structure (= PAGESIZE)
When defined, a logical page is the same size as PAGESIZE. To change the size of the logical page, specify PAGESIZE in the definition.

TOPMARGIN = length (= TOPMARGIN)
This is the top margin of the logical page. If it is not specified the current value of TOPMARGIN is used. If it is specified, it is added to the current value of TOPMARGIN.

BOTMARGIN = length (= BOTMARGIN)
This is the bottom margin of the logical page. If it is not specified the current value of BOTMARGIN is used. If it is specified, it is added to the current value of BOTMARGIN.

LEFTMARGIN = length (= LEFTMARGIN)
This is the left margin of the logical page. If it is not specified the current value of LEFTMARGIN is used. If it is specified, it is added to the current value of LEFTMARGIN.

RIGHTMARGIN = length (= RIGHTMARGIN)
This is the right margin of the logical page. If it is not specified the current value of RIGHTMARGIN is used. If it is specified, it is added to the current value of RIGHTMARGIN.

Assignments made to the following keywords during logical page definition *are ignored* because they have not been implemented.

LEFTGAPSTRING
RIGHTGAPSTRING
DEFCOLGAPSTRING

Any other commands encountered in the definition produce the error: *Not allowed in this type of definition. Ignored.*

Putting Text on the Logical Page

If &TWOCOL is the name of a logical page that has been defined, the command USE &TWOCOL causes text to be formatted on that logical page, rather than the physical page.

Several logical pages may be defined, but only one may be used on each physical page. To change to another logical page, give the USE command with the name of the new logical page. The command USE, without a name, returns to the physical page.

TEXTFORM sends a page to the output device when the page is full, or when a PAGEEND or PAGE command is issued. If you USE a logical page, and then USE the physical page, the logical page *will not appear if it was not ended*. Instead, TEXTFORM will generate it when it becomes full, or failing that, at the end of the run. To produce the logical page as soon as it is used, include PAGEEND before USEing the logical page, and return to the physical page with PAGEEND, USE PHYSICALPAGE. This technique is used in the following examples.

The COLUMN command begins the next column. This command has the same effect as PAGE if the current column is the last on the page, or if the page has only one column. In the command:

<**C**OLUMN [kwd] >

'kwd' can be used to specify the number of the column to start.

The COLUMNEND command ends the current column, if one was started. On the last column of a page, it has the same effect as the PAGEEND command.

<**C**OLUMN**END** [kwd] >

'kwd' describes how to vertically justify the text in the column, described on page 54.

TEXTFORM is unable to evenly balance the length of the columns across the logical page, if there is not enough text to fill each column. In some cases, your needs may be met by using the TABLE command. See the example on page 88.

TEXTFORM is unable to switch from columns to full width text and back to columns on the same page. However, you can place full width text at the top or bottom of a logical page by using the FLOAT command, as shown in the following examples.

Examples of Logical Pages

This example illustrates how to handle page numbers while using physical and logical pages.

```
<topmargin = 0in>
<define macro &topage, –
   lend, vs 2*linespace, pnctr, lend $thispage, –
edef macro &topage>
<reserve top page size default 1in &topage>
<define logicalpage &two, columns = 2, edef logicalpage &two>
<use &two>
<reserve top page size default 1in &topage>
<define macro &topcol, –
   'Column ', curcol, lend c, vs linespace, –
edef macro &topcol>
<reserve top column size default .5in &topcol>
<use> This text is on the physical page. <pageend, use &two>
<alignment = both>
To produce page numbers, the same RESERVE was specified twice —
once on the physical page, and once while the logical page was being
USEd. There is also a reserve for each column on the logical page.
<column> Footnotes<foot>Remember that footnotes use the
_d_e_f_a_u_l_t values of LINESPACE, TYPESIZE, ALIGNMENT, etc.,
so this footnote is not justified as is the rest of the text on the page.
<efoot> appear at the bottom of the current column.
```

produces:

1

This text is on the physical page.

2

Column 1

To produce page numbers, the same RESERVE was specified twice — once on the physical page, and once while the logical page was being USEd. There is also a reserve for each column on the logical page.

Column 2

Footnotes[1] appear at the bottom of the current column.

----

[1]Remember that footnotes use the *default* values of LINESPACE, TYPESIZE, ALIGNMENT, etc., so this footnote is not justified as is the rest of the text on the page.

The following example defines a three column logical page, and with a title above the three columns.

```
<define logicalpage &triple, –
  columns = 3, –
  defcolwidth = 9pica, –
  defcolgap = 2pica, –
enddefine logicalpage &triple>
<use &triple>
<def &ttl = 'A Three-Column Page With a Very Long Title<vs 4mm>'>
<float top page &ttl>
This logical page has three columns. When both DEFCOLWIDTH and
DEFCOLGAP are specified, TEXTFORM adds the remaining space to
LEFTGAP and RIGHTGAP.
<COLUMN> Notice how the title was FLOATed to the top of the
page, so that it could span more than one column.
<COLUMN> FLOATs can also appear at the tops of columns.
```

produces:

A Three-Column Page With a Very Long Title

This logical page has three columns. When both DEFCOLWIDTH and DEFCOLGAP are specified, TEXTFORM adds the remaining space to LEFTGAP and RIGHTGAP.

Notice how the title was FLOATed to the top of the page, so that it could span more than one column.

FLOATs can also appear at the tops of columns.

Logical Pages and the Default State

The first time you USE each logical page, the default state described on page 55 is in effect. If you want to change the defaults, do it *after* you USE the logical page. Any changes made remain in effect the next time the logical page is used.

Logical Pages With Reserves, Floats and Layouts

The physical page and each logical page have independent reserves and floats. If a reserve is in effect when a logical page is defined it does not automatically appear when the logical page is USEd. To produce a reserve on a logical page, USE the logical page and then issue the RESERVE command. This is illustrated in examples below. Remember than reserves, and floats, can be given for either pages or columns.

Logical pages may be defined and used with TEXTFORM layouts. To produce page numbers, *most* layouts set TOPMARGIN and BOTMARGIN to 0 inches, and then RESERVE 1 inch areas at the top and bottom of the page.

– If you define the logical page *before* you issue the LAYOUT command, all four margins will be 1 inch.
– If you define the logical page *after* you issue the LAYOUT command, TOPMARGIN and BOTMARGIN will be 0 inches for most layouts. You may want to respecify these values in the logical page definition.

In either case, the page numbers which are generated by the layout *will not* appear after you issue a USE name command. You will have to produce your own RESERVEs after you USE the logical page. To do this, see the example on page 79.

Information about Logical Pages

Information about the dimensions of a logical page is provided by using TABLE_INFO, as described on page 91, or by using the DISPLAY command as shown on page 92.

During a run, CURCOL contains the current column number. TEXTFORM changes it as soon as a column overflows, or is ended after a COLUMNEND. On the physical page, in page reserves and floats, and on a one column logical page,

CURCOL contains 0.

Tables

The INDENT command can achieve the effect of a table by moving to the next indent position on a line. Because this works only when the text does not overflow the line, TEXTFORM has a more flexible facility to handle tables.

There are three steps: 1) give the table dimensions by *defining* it, 2) enter the text into the table with the TAB command, which works the same way as the tab key on a typewriter (this step is called *filling* or *using* the table), 3) place the table on the formatted page (this is called *including* the table in the input). This empties the table, leaving it ready to be filled again. The following example produces a simple table, using these three steps.

```
<comment        Define the table>
<DEFINE TABLE &T1>
  <COLUMNS = 3>
<ENDDEFINE TABLE &T1>

<comment        Now put text in the table>
<USE &T1>
Column 1 <TAB 2> Column 2
<TAB 3> Column 3
<TAB 1> 11638 <TAB 2> 663404
<TAB 3> 93884
<TAB 1> 55672 <TAB 2> 557344 <TAB 3> 44725662
<TAB 2 3, SPLIT '–' >
<USE>

<comment        No longer putting text in table>
<comment        Include the table as input>
<&T1>
```

produces:

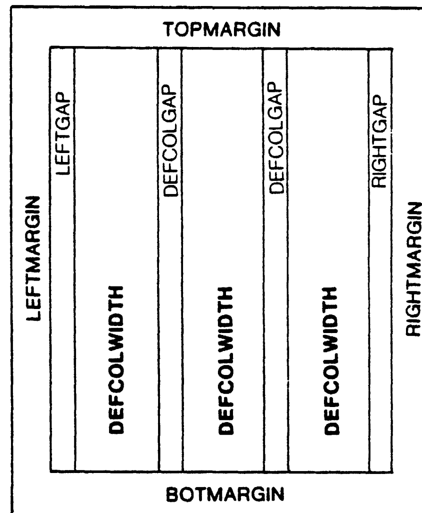
Column 1	Column 2	Column 3
11638	663404	93884
55672	557344	44725662

Defining the Table

When the DEFINE TABLE command is given, the table is built within the current column. If a logical page is being USEd, it is built to fit within the current column on the logical page. A table may be defined and then filled later in a document. If you often change PAGESIZE within a document, it is best to define a table near where it will be used, so that it is defined with the correct dimensions. A defined table does not change dimensions to fit the page on which it is used.

<**DEF**INE [kwd] **TA**BLE name>
keywords ...
<END**DEF**INE **TA**BLE name>

The keywords that appear within a table definition are the same as those for a logical page definition, on page 76. However, assignments should not be made to the following keywords in a table definition, since the result is incorrect:



PAGESIZE
LEFTMARGIN
RIGHTMARGIN
TOPMARGIN
BOTMARGIN

Putting Text in the Table

Text is placed in the table after the USE name command, where name is the name of the table. For example,

<USE &T1>

Within the table, the TAB command indicates the column, just as the tab key on a typewriter tabs across the columns. However, unlike a typewriter, once you have TABbed to a particular column, all text that you type before the next TAB command is kept in the tab column. This allows you to type multi-line entries for each tab column and let TEXTFORM fit them within the tab columns. TAB has the format:

<**T**AB [m [n]] >

If simply TAB, or T is given, all the text which follows appears in the *next* tab column. A *table entry* is the text between <TAB> and the next <TAB> command. If *m* is given, the text which follows appears in column *m*. If both *m* and *n* are given, as in TAB 2 3, text which follows is placed in a column that is the width of columns m through n (and all the formatting values used, such as font and alignment, are those of column *n*).

Since TAB starts a line in the next column, TAB, LINE produces the same result as just TAB.

Placing the Table on the Page

After the table has been filled, the USE command causes subsequent text to be included on the normal physical page. USE implies a LINEEND command as well. If &name is the name of a table, as &T1, <&name> ends the current line, causes the table to be placed as text on the page, and then empties the table. It may then be filled again. If you need to print the same table with the same contents more than once, put the commands to fill and include it in a macro, and use the macro whenever

you need the table. <&name = ' '> also empties the table, without including it as input.

Extra Vertical Space In Tables

If you want extra vertical space to appear across all columns, give these commands to insert one blank line:

```
<TAB 1, BC, TAB 1>
```

The following macro lets you insert a variable amount of vertical space:

```
<DEF MA &TSEP, –
   T 1, –
   IF NRPARS=0 THEN, BC, ELSE, VS PAR(1), ENDIF, –
   T 1, –
EDEF MA &TSEP>
```

For example, &tsep(6po). The following two sequences are usually incorrect:

```
<TAB 3>text <VERTSPACE 4MM, TAB 1>
```

inserts the vertical space in column 3 only, while

```
<TAB 1,VERTSPACE 4MM>text
```

inserts the vertical space in column 1 only.

To end a page in tabular material, for example in table &TAB, use the commands:

```
<USE, &TAB, PAGEEND, USE &TAB>
```

To end a page in tabular material which is being placed on a logical page, use the commands:

```
<USE &logicalpage, &TAB, PAGEEND, USE &TAB>
```

PAGE, COLUMN, PAGEEND, or COLUMNEND commands cannot appear while a table is being USEd.

Error Messages Related to Tables

If the specified number of columns defined cannot fit across the page, an error is issued: *xxxx more than the width of the logical page or table has been assigned. Automatic sizing invoked*

If the space for text in a table column is too narrow, this may produce the error *Unable to keep non-line-breaking words within the margins.* If no indents are in effect, you can define the specific column to be wider than other columns.

The TAB command is valid only when a table is being USEd. At any other time, it causes the error *The TAB command is only valid inside tables. Ignored.* An attempt to tab past the last column in the table produces the error *Attempt to TAB past last column of table. Tabbing to Column 1.* When you start to enter text in a table with the USE command, you are positioned at TAB 1, so a TAB 1 command is redundant at that point.

A table entry must fit on one column of a page. If it does not, TEXTFORM ends the run with the error *Table entry is too long to fit on the page. Terminating.* However, a complete table may span several pages, as long as the individual entries are each less than a page (or a column on a page).

TEXTFORM will not split a table entry across two pages, so the KEEP command is unnecessary. PAGE, COLUMN, PAGEEND, or COLUMNEND commands cannot appear while a table is being USEd. These commands produce the error *PAGE, COLUMN, PAGEEND and COLUMNEND commands are not allowed in tables. Command ignored.*

When a table is being used, do not issue FOOTNOTE, FLOAT, VERTJUST, or KEEP commands. To produce a footnote, enter only the superscript number in the table (for example <SUP1>) and then type the same number and the contents of the footnote *as regular text after the table has been placed on the page*.

Tables and the Default State

1) When defining the table, the values for font and alignment in each column are the current values, unless these are changed by a DEFINE COLUMN command. If an indent is in effect while the table is being defined or filled, the indent is saved, set to zero, and restored after the definition.

2) When putting text in the table, with a USE &table command, TEXTFORM stores the values in the default state list (described on page 55). When a USE command is given to stop putting text in the table, these values are restored. This means that any changes to these values inside the table don't affect text which follows the table.

Inside the table, any font or alignment changes affect only the column in which the change occurs. These changes remain in effect throughout the column, unless they are reset. They also remain in effect for that column if a table is placed on the page, and then USEd again.

In contrast, indent commands affect not only the current column, but subsequent columns, until the indent is reset. In a column, the indent is from the left of the column. If the space for text remaining in the column is too narrow for text, it may produce the error *Unable to keep non-line-breaking words within the margins.* See the DEFINE COLUMN command on page 91 to define a column of a specific width.

3) When placing the table on the page, the table will be the full width it was when defined. If an indent is in effect, the table may overflow the margins.

Examples of Tables

The first example specifies the physical dimensions of the table. In this manual, examples are indented from the left margin. When the table is placed on the page, this indent appears, then the LEFTGAP of .5 inch, and then the table. If the table had been defined to be a full 6.5 inches, and was placed on the page while an indent was in effect, it would overflow into the right margin.

```
<def table &t4, –
  defcolwidth=1in, –
  defcolgap=0in, –
  leftgap=.5in, –
  rightgap=.5in, –
  columns=4, –
edef table &t4>
<use &t4>
23<split '.', tab 2>66<split '.'>
<tab 3>27<split '.', tab 4>47<split '.'>
<tab 1>55<split '.', tab 2>87<split '.'>
<tab 3>64<split '.', tab 4>51<split '.'>
<tab 1>46<split '.', tab 2>96<split '.'>
<tab 3 4>98<split '.'>88
<use, &t4>
```

produces:

```
23.................... 66.................... 27.................... 47....................
55.................... 87.................... 64.................... 51....................
46.................... 96.................... 98......................................... 88
```

This example uses the DEFINE COLUMN command to describe each column in the table. For more information about this command, see page 91.

```
<define table &t5, –
  defcolwidth=30mm, –
  defcolgap=5mm, –
  columns=4, –
  define column 1, –
    alignment=right, edef column 1, –
  define column 2, –
    alignment=centre, edef column 2, –
  define column 3, –
    alignment=left, edef column 3, –
  define column 4, –
    width = textwidth('Totals'), –
    edef column 4, –
edef table &t5, –
leftindents=(.3in), –
use &t5> 636 <tab 2> 1832 <tab 3> A. <indent left 1> Initial rate of
de<->cline <tab 4,i l 0> Totals
<tab 1, bc, tab 1> 80655 <tab 2>199730 <tab 3, indent left 0>
B. <indent left 1>Percen<->tage decrease <tab 4,i l 0> Totals
<use,&t5>
```

produces:

636	1832	A.	Initial rate of decline	Totals
80655	199730	B.	Percentage decrease	Totals

The next example defines a table when indents are in effect (perhaps as the result of a layout). The LEFTGAP of the table is set to the value of the current left indent, so that the first column of the table aligns with the text above and below it.

```
This is some indented text. The table will be defined so that it aligns
with this text.
<define &lin = lindent>
<define &lindx = lindentindex>
<define table &t6, –
   columns = 5, –
   leftgap = &lin, –
edef table &t6>
<use &t6>
<t 1> aaaa <t> dddd <t>123 <t>123 <t>000
<t 1> bbbb <t> eeee <t>456 <t>123 <t>000
<t 1> cccc <t> ffff <t>789 <t>123 <t>000
<use, &t6, indent left &lindx>
Now reset the indent that was saved and continue the regular text.
```

produces:

This is some indented text. The table will be defined so that it aligns with this text.

aaaa	dddd	123	123	000
bbbb	eeee	456	123	000
cccc	ffff	789	123	000

Now reset the indent that was saved and continue the regular text.

When the table is described only by the number of columns it contains, TEXTFORM calculates the width of columns and gaps. In the following example, columns 1 through 11 are defined; the last column will have all the remaining space in the table. This may be useful if you know the required width of certain columns, and want the rest of the space to be spread among the remaining columns.

```
<def table &t7, columns=12, –
leftgap=.5IN, defcolgap=0, –
def col 1, width=1.2IN, edef col 1, –
def col 2, width=.2IN, edef col 2, –
def col 3, width=.2IN, edef col 3, –
def col 4, width=.2IN, edef col 4, –
def col 5, width=.2IN, edef col 5, –
def col 6, width=.2IN, edef col 6, –
def col 7, width=.2IN, edef col 7, –
def col 8, width=.2IN, edef col 8, –
def col 9, width=.2IN, edef col 9, –
```

```
def col 10, width=.2IN, edef col 10, –
def col 11, width=.1IN, edef col 11, –
edef table &t7>
<use &t7> Card <t 8> 1 <t>1 <t> 1 <t 12> Values read into TSP
<t 1> Column <t>1 <t>2 <T>3 <t>4 <t>5 <t>6 <t>7 <t>8 <t>9
<t 6>1 <t>2 <t>3 <t>6 <t>7 <t 12> 12367.32
<t 2>1 <t>2 <t 12>1200000000.00
<t 9>1 <t>2 <t 12>.12
<t 7>1 <t>2 <t>0 <t>0 <t 12>12200.00
<use, &t7>
```

produces:

Card							1	1	1	Values read into TSP
Column	1	2	3	4	5	6	7	8	9	
					1	2	3	6	7	12367.32
	1	2								1200000000.00
								1	2	.12
						1	2	0	0	12200.00

The following example uses the table facility to give the appearance of multi-column text. Tables are used so that the two columns can be balanced throughout the page; this cannot be done with logical pages. The table is placed on the page after each row of entries so that the table entries do not become too large. Remember that the entry must fit on one column of a page. Note that the TAB 1 command is unnecessary immediately after the USE command:

```
<define table &bylaw, –
  columns=2, –
  leftgap=0.5409IN,–
  define column 1, –
    alignment=left, –
    width=1in, font 2 –
  edef column 1, –
  define column 2, –
    alignment=both, –
    width=3in, –
  edef column 2, –
edef table &bylaw>
<FONT 3>Bridge Regulations<FONT 1, LEND C, VS 4MM>
<USE &BYLAW> 804 <LEND> Traffic Regulations
<TAB> (1) No person shall operate any vehicle which is steered or
controlled by a system of levers nor any bulldozer with a blade upon
the roadway traffic deck of the High Level Bridge.
<USE, &BYLAW, LEND, USE &BYLAW>
804.1<TAB> (2) No person shall park a vehicle upon the roadway
traffic deck of the High Level Bridge.
<USE, &BYLAW>
```

produces:

Bridge Regulations

804 *Traffic* *Regulations*	(1) No person shall operate any vehicle which is steered or controlled by a system of levers nor any bulldozer with a blade upon the roadway traffic deck of the High Level Bridge.
804.1	(2) No person shall park a vehicle upon the roadway traffic deck of the High Level Bridge.

Putting Text in a Table by Row or Column

You can change an entry in the table any time after table definition, or after the table has been filled.

```
<define table &change, columns = 4, enddefine table &change>
<use &change, 'aaa', tab 2, 'bbb'>
<tab 3, 'ccc', tab 4, 'ddd'>
<use, &change(2,1) = 'x', &change>
```

These commands produce the message *A table entry with this index (1,1) has already been inserted. It will be deleted.* The following table is produced:

aaa	X	ccc	ddd

This method also lets you fill a table by columns rather than by rows.

```
<&change(3,1) = 'column 3'>
<&change(2,1) = 'column 2'>
<&change(1,1) = 'column 1'>
<&change>
```

produces:

column 1	column 2	column 3

Tables with Reserves, Floats and Logical Pages

If you want to produce text within reserves in the same table format as text on the page, define two tables with similar dimensions, and use the second table in the reserve. Do not use the same table for both open text and reserves.

If a table is defined on a logical page, *after text has appeared on the logical page,* the width of the table is determined by the width of the column. However, the following commands are not enough to cause TEXTFORM to define a table the width of one column:

```
<DEFINE LP &2COL, COLUMNS=2, EDEF LP &2COL>
<USE &2COL>
<DEFINE TABLE &T, . . .
```

This is because the USE command does not cause TEXTFORM to adjust CURPAGESIZE. This is done only after text is placed on the logical page. The best procedure is to start the column with a COLUMN command, or a title for the table, and then define the table.

When including the table on a logical page, a common mistake is to say USE,&TABLE, which will put the table on the PHYSICALPAGE instead of the logical page. If the logical page is &2COL, and the table is &TABLE, do the following:

```
<USE &2COL>
This text appears in the column.
<USE &TABLE>
This text fills the table.
<USE &2COL>
<comment now back on the column>
<&TABLE>
<comment include the table>
```

If you do not know the name of the logical page, you can save it as follows (use of the $ operator is described on page 108):

```
<DEF &SAVELP = CURLP>
<USE &TABLE>
This text fills the table.
<USE $&SAVELP, &TABLE>
```

If you require a table in a float on a logical page, for example &2COL, your float should be treated as follows:

```
<use &2col>
this text is on the two column page. Now we need a bottom PAGE
float that contains a table.
<define macro &fltab>
  <comment define a table>
  <use &table> <comment fill the table>
  <use &2col> <&table>
  <comment the table will be the width of the PAGE>
<enddefine macro &fltab>
<float bottom page size default 3in &fltab>
```

If we use a float bottom column command, the float will be the width of the column.

Information about Tables

CURTABLINE contains the number of the current line in the table. Outside of a table it is 0.

```
<define table &t1>
<columns = 5>
<edef table &t1>
<use &t1>aaa <curtabline>
```

```
<tab 1>bbb <curtabline>
<nl>ccc <curtabline>
<use, &t1>
```

produces:

```
aaa 1
bbb 2
ccc 2
```

TABLE_INFO can be used to get information about the dimensions of a table, or logical page. The following example stores a length in LEFTINDENTS(30) that is the width of column 1 and the gap between column 1 and column 2 of the table.

```
<define table &tab, –
  columns=6, –
  define column 1, font 3, edef column 1,–
edef table &tab>

<def &wwid = table_info(&tab, 'width',1)>
<&wwid = &wwid+table_info(&tab, 'gap',2)>
<leftindents(30)=&wwid>
```

Information about the dimensions of table can also be displayed by using the DISPLAY command as shown on page 92.

Defining Specific Columns in Logical Pages and Tables

In a logical page or table definition, any number of columns can be defined. Note that some of the keywords are valid only when the column is being defined inside a table.

<**DEF**INE [kwd] **COL**UMN expression>
keywords ...
<**END****DEF**INE **COL**UMN expression>

kwd
may be one of the AXR or NXR. See cross references on page 176.

expression
is the number of the column being defined. It must be 1 or greater.

keywords
GAP = length (= DEFCOLGAP)
This is the gap between the preceding column and the column. Column 1 may not have a GAP (use LEFTGAP). If GAP is not specified, or is –1, its width is determined after all the columns in the logical page or table are defined.

WIDTH = length (= DEFCOLWIDTH)
This is the width of the column. If not specified, it is determined by TEXTFORM after all the columns in the logical page or table are defined. To make the width of the column wide enough for a specific word, use TEXTWIDTH:

```
<WIDTH = TEXTWIDTH('Specifications  ')>
```

If the total width of the columns in a logical page or table is too wide, the following error appears: *xxxx more than the width of the logical page or table has been assigned. Automatic sizing invoked.*

ALIGNMENT = kwd (= current value of ALIGNMENT)
This is the alignment of the text in the column. If not specified, the current value of ALIGNMENT is used. This command is only valid in COLUMN definitions for tables.

FONT [expression] (= CURFONT)
Text in the column will appear in this font. This command is only valid in COLUMN definitions for tables.

COMMENT body
A comment may appear.

Displaying the Logical Page or Table Dimensions

If LIST SOURCE (see page 172) is in effect, the characteristics of the logical page or table are displayed in the listing, using DEFUNITS as the unit of length. This same information can be displayed on SERCOM, as in the following example.

```
<pagesize = (8.5in, 11in)>
<leftmargin=1in, rightmargin=1in>
<topmargin=1in, botmargin=1in>
<define logicalpage &twocol, –
   columns = 2, –
   defcolgap = .5in, –
edef logicalpage &twocol>

<defunits=in, display &twocol>
```

produces:

```
ORIGIN=(1IN,1IN)  PAGESIZE=(7.5IN,10IN)
LEFTMARGIN=1IN  RIGHTMARGIN=0IN  TOPMARGIN=1IN
BOTMARGIN=0IN
LEFTGAP=0IN  LEFTGAPSTRING=
RIGHTGAP=0IN  RIGHTGAPSTRING=
Column  Gap        Width        Gapstring
1                  3IN
2       0.5IN      3IN
```

Use the displayed information to confirm the width of the columns, the gap between columns, and the left and right gap. You can also display variables, but cannot display logical page or table keywords such as COLUMNS, etc. However, you can use

TABLE_INFO to store information about logical page or table keywords in variables which you can then display.

In the following example, TEXTFORM calculates the widths of the columns and gaps; the gaps are set to 10% of the column width needed to fill the page:

```
<define  logicalpage  &3col,  –
   columns  =  3,  –
   leftgap  =  –1,  rightgap  =  –1,  –
enddefine  logicalpage  &3col>
<defunits=in,  display  &3col>
```

produces:

```
ORIGIN=(1IN,1IN)  PAGESIZE=(7.5IN,10IN)
LEFTMARGIN=1IN  RIGHTMARGIN=0IN  TOPMARGIN=1IN
BOTMARGIN=0IN
LEFTGAP=0.1912IN  LEFTGAPSTRING=
RIGHTGAP=0.1912IN  RIGHTGAPSTRING=
Column  Gap          Width          Gapstring
1                    1.9118IN
2       0.1912IN     1.9118IN
3       0.1912IN     1.9118IN
```

TEXTFORM spreads the three columns across 6.5 inches (PAGESIZE(1)–LEFTMARGIN–RIGHTMARGIN) allowing for the width of the columns (3×1.9118), the width of LEFTGAP and RIGHTGAP ($2 \times .1912$), and the width of DEFCOLGAP ($2 \times .1912$). The gaps are all set to 10% of the column width.

Producing Columns Without Using DEFINE TABLE

TEXTFORM allows three different modes for incoming text (UNFORMATTED, ASIS, and PREFORMATTED). Two of these modes can be used to produce tabular material, in certain cases only.

The default is UNFORMATTED mode, where TEXTFORM makes all formatting decisions. This mode is in effect at the beginning of a TEXTFORM run. In this mode, all multiple occurrences of blanks are reduced to a single blank. TEXTFORM attempts to put as many words onto a line as possible without overflowing the margins. All TEXTFORM commands are honoured and acted upon. This mode is produced with the command:

```
<INPUTMODE=UNFORMATTED>
```

When it is desirable to enter text exactly as it is to appear on the page, but still have commands acted upon, use PREFORMATTED. This mode is used only when the characters on the intended output device are the same width. Return to normal processing by setting INPUTMODE back to UNFORMATTED:

```
<INPUTMODE=PREFORMATTED>
=  *  *  <CAP  =  TRUE>  =
aaaaa          bbbbb
```

```
<INPUTMODE=UNFORMATTED>
```

produces:

```
= * *  =
AAAAA        BBBBB
```

PREFORMATTED mode uses the alignment specified explicitly on LINEEND commands, but it does not use the value of ALIGNMENT at any time. Blank lines in the input produce blank lines in the output. Command lines do not produce blank lines in the output unless blanks appear around the command.

The third mode of input, ASIS, also attempts to duplicate the input data on the output device, but commands and meta-characters (¬ @ _) are *not* honoured. Like PREFORMATTED, ASIS mode is useful only if the intended output device has characters which are all the same width (as is the case on the line printer). Otherwise the output will not closely resemble the input. To end ASIS mode and return to normal processing, enter the commands:

```
<INPUTMODE=UNFORMATTED>    or
  <INPUTMODE=PREFORMATTED>
```

in column 1 of the input. *Blanks may not appear around the equal sign in these commands.* For example:

```
<INPUTMODE=ASIS>
= ¬ ¬ <CAP = TRUE> =
aaaaa        bbbbb
<INPUTMODE=UNFORMATTED>
```

produces:

```
= ¬ ¬ <CAP = TRUE> =
aaaaa        bbbbb
```

Within ASIS and PREFORMATTED mode, when the width of the characters on one input line exceeds the allowed width of an output line:

– the line is broken if it contains blank spaces, and the subsequent portion of the line appears on the next line. This causes the error: *Line width exceeded in ASIS or PREFORMATTED mode.*
– the line will extend into the margins if it contains no blank spaces, producing the errors *Line width exceeded in ASIS or PREFORMATTED mode. Unable to keep non-line-break words within the margins.*

Note that the use of PREFORMATTED and ASIS modes prevents a document from being completely device independent, and also makes editing extremely difficult, since spaces are important. Whenever possible, use the indent, table, or column facilities rather than ASIS or PREFORMATTED modes. Do not use AT points when INPUTMODE is ASIS, because commands are not honoured.

VARIABLES

Predefined Variables

Several variables such as LINESPACE, PAGESIZE, and ALIGNMENT have already been described in this manual. Because they are available at the beginning of a TEXTFORM run, and contain default values, these variables are called **predefined** variables. You may change the values of predefined variables, unless the variable becomes constant during the run.

System Variables

Throughout this manual there have been references to variables in TEXTFORM which change to convey information about the current state. You may check the value of these variables, but not change them yourself. These are called **system variables**. For example, THISPAGE is a system variable which contains LEFT or RIGHT. See the Index to TEXTFORM Language for a complete list of system variables.

User-Defined Variables

You may define and use your own variables with the DEFINE command:

<**DEF**INE name [= expression] >

For example:

```
<DEFINE &EF>
<DEFINE &EFA = 'Encrinurus (Frammia) articus'>
```

In the DEFINE command, ‘expression’ is the initial value to be assigned to the variable. If ‘expression’ is omitted, as in DEFINE &EF, the variable is given a **null** string as its value. A null string is '', which means it has no value, or is empty.

To use a variable, enter its name as a TEXTFORM command in the text. When TEXTFORM encounters it, the variable is replaced by its value:

```
<define &long = "a long phrase I'm tired of typing">
Now when I come to <&long>, I put in the variable name.
```

produces:

> Now when I come to a long phrase I'm tired of typing, I put in the variable name.

If you have values (numbers or strings) which are likely to change in the future, those values should be placed in variables. It is also a good practice to place these variable definitions at the beginning of a source file. When you need to change these values later, they will be easier to find. In this way the document can be reformatted with a minimum amount of effort.

A variable may contain TEXTFORM names. This includes commands or other variable names:

```
<DEFINE &2 = 'Text and <F 2>TEXTFORM commands<F>'>
<DEFINE &ET = '<FONT 2>et¬al<F>'>
```

The contents of a variable may be changed by assigning a new value,or by making it null:

```
<&ET = 'Encrinurus (Frammia) articus'>
<&ET = ' ' >
```

Choose names for your variables that are short (to save typing) but meaningful (in case you have a great number of them).

Variables may be created that contain a group of often repeated TEXTFORM commands. Here is an example that uses the INDENT command to create a bibliography.

```
<DEF &BIB = '<LEND, I OFF, VS 4MM, I H=1 L 3>'>

<&BIB> 1946. U.S. Army Ordnance Corps.
"Mathematics by Robot," <U>Army Ordnance<U OFF>,
Vol. XXX, No. 156 (May-June), pp. 239-331.

<&BIB> 1961. Weik, Martin H. "The ENIAC Story,"
<F 2>Army Ordnance<F>, Vol. XLV, No. 244
(January-February), pp. 571-575.
```

produces:

1946. U.S. Army Ordnance Corps. "Mathematics by Robot," Army Ordnance, Vol. XXX, No. 156 (May-June), pp. 239-331.

1961. Weik, Martin H. "The ENIAC Story," *Army Ordnance*, Vol. XLV, No. 244 (January-February), pp. 571-575.

Values that can be Assigned to Variables

Several types of items, described below, may be assigned to a variable. They are:

string
number (integer, scaled, or length)
logical value
pointer
hex
structure

Strings

Strings are specified as characters between single (') or double (") paired quotes. The single or double quote is used as a *delimiter*.

To assign the string ‘Chapter 1’ to the variable &COUNT, the command is:

```
<&COUNT = 'Chapter 1'> or <&COUNT = "Chapter 1">
```

A variable may be changed to contain a string even though the variable may previously have contained another type of data item. Both delimiters must appear on a string. The command:

```
<&COUNT = 'Chapter >
```

produces *String delimiter missing.*

The character being used as a string delimiter may be used in the string by entering the character twice:

```
<&COUNT = 'It''s not confusing'>
```

or by using the alternate delimiter around the string:

```
<&COUNT = "Let's quit">
```

There can be several levels of command mode within strings, as long as you can keep them straight:

```
c<'o<"n""f'>us''in'>g
```

produces:

```
con"fus'ing
```

The variable length cannot exceed 256 characters if it is going to be included in the input. A variable longer than this can be displayed, but if it is included in the input it causes the error: *Input line longer than 256 characters. Truncated.* To include a long variable in the input, use substringing, below.

CHARS, TEXTONLY and SEPARATE are provided to manipulate portions of strings. SEPARATE is described on page 107. CHARS determines the number of

characters in a string:

```
<CHARS('abcdef')>
```

produces 6.

You can force the first character of a string to be capitalized as follows:

```
<DEFINE &AA = 'title' >
< '@':&aa >
```

produces:

```
Title
```

TEXTONLY removes commands from strings, and leaves only special character names.

```
<DEFINE &ALL = 'This is an <FONT 2>emphasized<F> <integral>.'>
<DEFINE &ALLOUT = TEXTONLY(&ALL) >
```

makes &ALLOUT:

```
‘This is an emphasized <integral>.’
```

REP repeats a string of characters in the input.

```
<FOOTSEP = '<REP(18,''-''),LEND>'>
```

produces the footnote separator:

```
------------------
```

Substrings

A **substring** is described as a ‘name’ followed by a substring descriptor; (a parenthesised pair of ‘expression’s separated by a substring operator).

– Strings have a 1 origin index—the first character of the string has an index of one, the second an index of two, etc. Neither ‘expression’ may be zero or negative.
– If the value of ‘name’ is not a string, it is copied and converted to a string for the substring operation.
– If the value of ‘name’ is not long enough to contain the whole substring being requested, the substring returned will contain all that it can from ‘name’. The substring will then be padded out to the requested length with blanks.

Substringing is available in two forms; as a from-to pair, or as a from-length pair. This requires two substring operators:

Variables

from-to expression1 ; expression2

The substring of ‘name’ returned is from ‘expression1’ to ‘expression2’. If &A is 'abcdefghij' then:

<&A(1;5)> is 'abcde', and
<&A(3;6)> is 'cdef'.

from-length expression1 | expression2

The substring of ‘name’ returned is from ‘expression1’ for a length of ‘expression2’. If &A is 'abcdefghij' then:

<&A(1|5)> is 'abcde', and
<&A(3|6)> is 'cdefgh'.

Note that when ‘expression1’ is 1, both ; and | produce the same result.

If a name contains more than 256 characters, it cannot be directly included in the input. Substringing can be used in include the contents of the name in parts. In the following example, assume the string is less than 512 characters:

< &name(1;255), &name(256;chars(&name)) >

Numbers

Numbers may be integer, such as 1, 4, 212, 1000000. They may have negative values. The largest and smallest numbers are:

2147483647
-2147483648

Mathematical operations may be done on numbers, as described on page 110:

<DEFINE &CNT=1, &CNT=&CNT+1>

Scaled Numbers

Scaled numbers may have up to four decimal places, such as 4.8856, and may contain negative values. If you try to provide more than four decimal places, as in:

<&COUNT = 1.23456>

produces the error *Number truncated to 4 decimal places.*

The largest and smallest scaled numbers are:

214748.3647
-214748.3648

Lengths

A length is entered either as a number or scaled number followed by a length indicator, one of:

- MILLIMETRE, MILLIMETER, MILLIMETRES, MILLIMETERS or MM
- INCH, INCHES or IN
- PICA, PICAS or PI
- POINT, POINTS or PO
- LINE, LINES or LI – the smallest amount the device can move in a vertical direction; should be used only for vertical directions
- UNIT, UNITS or UN – the smallest amount the device can move in a horizontal direction, and should be used only for horizontal directions. Units depend on typesize, so that 30 units in 6 point type are narrower than 30 units in 18 point type.

Lengths are stored internally in 100ths of a micron. Generally, lengths are not negative, but negative lengths are allowed with the VERTSPACE, BLANKCHARACTER and HORSPACE commands. As operands for these commands, a negative length implies direction (towards the top of the page, or towards the left margin respectively).

Although lengths are stored internally in 100ths of a micron, the smallest usable length is actually governed by the output device in use. For example, the smallest horizontal space on the 1403 line printer is 1/10 of an inch and the smallest vertical space is 1/6 of an inch. The largest length that TEXTFORM can store in a variable is 845.4663 inches.

If a length is expected and no unit of length is given, the value contained in the variable DEFUNITS is used as the length.

The following errors may appear when assigning lengths:

```
<&A = 1IN * 3IN>
```

produces *Both arguments of multiplication have type LENGTH.*

```
<&A = 1 + 3IN>
```

produces *Only one argument of addition or subtraction has type LENGTH.* However, this rule is relaxed when 0 is used in the addition or subtraction.

The values of INCHES, MILLIMETERS, POINTS and PICA are device independent—they can be used on any output device, although the device may not be able to produce the exact length specified. The following table shows the relationship between these lengths:

	IN	**MM**	**PI**	**PO**
IN	1	25.4	6.025	72.3008
MM	0.0394	1	0.2362	2.8346
PI	0.1667	4.2333	1	11.9999
PO	0.0139	0.3528	0.0833	1

The values of UNIT and LINE are device units. A UNIT represents the smallest

horizontal movement the output device is capable of moving; the value may depend on the current character size, or typesize. A LINE is the smallest vertical movement the output device is capable of moving. You should be familiar with the output device being used if you are going to give lengths in UNITS or LINES. On the line printer, the size of a unit is the same width as a letter. On the phototypesetter, its size depends on the typesize being used.

Since a line is the smallest vertical space the device can move, this too may change according to the device and character set being used. It is always 1/6 of an inch on the line printer, but varies according to the character set on some other devices.

Units and lines are very device dependent and should be used sparingly. Because they make your document dependent upon one output device, lengths specified in units for one device may produce disastrous results on another device.

Determining the Width (Length) of a String

TEXTWIDTH is used to determine the length of one or more characters:

```
<TEXTWIDTH(  expression  )>
```

The value returned is device-dependent, because it depends on the current character set, typesize, and font:

```
<defunits  =  units>
<od  '1403'  'tn'>
<define  &wid  =  textwidth('123')>
```

makes &WID=3units, while:

```
<defunits  =  units>
<od  'aps5'  'geneva'>
<typesize  10points>
<define  &wid  =  textwidth('123')>
```

makes &WID=168units.

If you include commands within the string, the commands are not evaluated. The result returned contains the width of all characters. Blanks are treated as being WORDSPACE wide; when consecutive blanks are encountered in the string, the value of one WORDSPACE is used:

```
<DEFUNITS=IN,  DEF  &WID  =  TEXTWIDTH('word       in  <FONT  2>')>
```

makes &WID=1.6IN if the character set is TN. It is convenient to set indents based on the width of characters. However, note that:

```
<DEFINE  &1  =  '<TEXTWIDTH(''abc'')>'>
```

makes &1 a string, instead of a length, so that:

```
<LEFTINDENTS(1)  =  &1  >
```

will *not* work. Instead, make &1 a length with:

```
<DEFINE  &1  =  TEXTWIDTH('abc')>
```

or use:

```
<LEFTINDENTS(1)  =  TEXTWIDTH('abc')  >
```

to produce a length. You can also use the name of a variable with TEXTWIDTH, or use the length calculated by TEXTWIDTH in other commands:

```
<define  &footer  =  'This  book'  >
<blankcharacter  textwidth(  &footer  )>
```

Logical Values

Logical values may be specified as 0, OFF, NO, FALSE; or 1, ON, YES, or TRUE. An example of a variable which contains a logical value is ODLOADED. The result of any comparison is either TRUE or FALSE.

Pointers

Pointers point to some other name. Pointers are entered as an @ followed by the name you wish to point to (e.g., @&B is a pointer to &B). See details about pointers on page 108.

Hex

Hexadecimal information is entered as an even number of hexadecimal digits delimited by the hexadecimal delimiter #.

```
Input  in  <#C885A7#>.
```

produces:

```
Input  in  Hex.
```

If you assign a hex value to a variable, it becomes a NUMBER. Note that numbers may only have a length of 1, 2, 3, or 4 bytes (2, 4, 6, or 8 hexadecimal digits).

If you use a hex value where a string is required, and catenate it to a null string, it becomes a string: #5C#:'' becomes a *. This can be used in the AT TEXTCHARACTER command for hex values that do not have TEXTFORM special character names.

Structures

Structures are a special form of variable—a parenthesized list of values separated by command separators. Each value in the structure is called an **element** of the structure, and contains the same types of information that a variable contains. The elements must be enclosed in parentheses, even if there is only one element. In TEXTFORM, structures are defined like variables, and the name has the same form as a variable name:

```
<DEFINE &LIST = ( 2, 4, 6 )>
```

defines a structure with three elements, or parts. The types of the structure elements need not be the same. For example:

```
<DEFINE &STUFF = ( 36, 'some text', 3IN )>
```

is a valid structure.

The structure may be treated as a whole or by the individual parts:

```
<&STUFF>
```

produces:

```
36some text3IN
```

while:

```
<&STUFF(3)>
```

produces:

```
3IN
```

If you need to use indents and don't want to change those already in effect, change an un-used indent:

```
<LEFTINDENTS(23) = 5IN>
```

which will not modify any indent other than the twenty-third indent. When you refer to part of a structure, as in LEFTINDENTS(23), you are **indexing** the structure. An index of 23 refers to the twenty-third element in the structure. An index of 0 refers to the entire structure, so &STUFF(0) is the same as &STUFF.

Mathematical operations can be performed on the structure. If &LIST = (2, 4, 6), then:

```
<&LIST = &LIST * 2>
```

makes &LIST equal to:

```
( 4, 8, 12 )
```

while:

```
<&LIST = &LIST:'abc'>
```

makes &LIST equal to:

```
( '2abc', '4abc', '6abc' )
```

Operations can also be done on parts of the structure. If &LIST = (2, 4, 6) then:

```
<&LIST(2) = &LIST(2)+100>
```

makes the structure:

```
( 2, 104, 6 )
```

Notice where the parentheses appear when you define a one element structure, such as LEFTINDENTS=(1IN), and when you change only the *first* element of the structure, with LEFTINDENTS(1)=1IN. In this manual, several structures have already been discussed. These are PAGESIZE, LEFTINDENTS, and RIGHTINDENTS. PAGESIZE is initially (8.5IN, 11IN). If you want to reduce the size by one half, you could type either:

```
<PAGESIZE = ( 4.25IN, 5.5IN )>
```

or:

```
<PAGESIZE = PAGESIZE / 2 >
```

or:

```
<PAGESIZE = PAGESIZE * .5>
```

A variable can be turned into a structure by subscripting its name.

```
<&A(3) = 11>
```

will make &A a structure (if it isn't already), then make its third element 11. Note that if &A is not a structure, its current value is *not* included as the first element of the new &A. If it is a structure and doesn't have three elements, TEXTFORM makes it have three elements (those without values are made into null strings):

```
<DEFINE &SUM = 5>
<&SUM(3) = 11>
```

makes &SUM equal to (, , 11). It is *not* (5, ,11). Structures and pointers are often used together. For example:

```
<DEFINE &CTR = 1>
<DEFINE &TITLE = ('Chapter ', @&CTR, '. ')>
<&TITLE>
```

produces:

```
Chapter 1.
```

If the value of &CTR changes, so does the value of &TITLE:

```
<&CTR=3, &TITLE>
```

produces:

```
Chapter 3.
```

You could also change &CTR in this way:

```
<&TITLE(2)=5, &TITLE>
```

produces:

```
Chapter 5.
```

Information about Variables and Structures

VTYPE tells you the type of a variable; it produces a string that is a multiple of 16 characters long.

```
<DEFINE &TEST = @LINESPACE>
<VTYPE(&TEST)>
```

produces:

```
POINTER
```

VECLEN tells you the the number of elements in a structure.

```
<DEFINE &LEN = (2, 4, 'abc', @&D)>
<VECLEN(&LEN)>
```

produces:

```
4
```

MEMBER tells you the index of a specified element in a structure. To use it, give the element you are looking for, followed by the structure. TEXTFORM then scans the structure for the element. If the element is found, its index is returned; otherwise the result is zero.

```
<MEMBER( 'abc', ( 34, 'thing', 'abc', 211) ) >
```

produces:

```
3
```

SUBSTRUC produces a substructure of a given structure. To use it, give the name of the initial structure and two substructure indices.

```
<DEFINE &STRUC = ('A','B','C','D')>
<DEFINE &SUB = SUBSTRUC(&STRUC,2,3)>
```

The resulting structure &SUB is:

```
('B','C')
```

STRUCTURE catenates two or three structures into a new structure.

```
<DEF &A = ('A','B'), DEF &B = (1,2) >
<DEF &C = STRUCTURE( &A, &B )>
```

The resulting structure &C is:

```
('A','B',1,2)
```

STRUC forms a structure.

```
<define &len = 1in>
<define &s = struc(('abc', 123, &len ))>
```

produces the structure &S:

```
('abc', 123, 25.4MILLIMETRES)
```

MAX and MIN scan through a structure and find the maximum or minimum value. To use them, give the type of comparison desired within string delimiters (length, string, number) followed by the structure or structure name. For instance, if you indicate a comparison as ‘LENGTH’, then, every element in the structure is treated as lengths. Conversion is done automatically if the element was not given as a length.

```
<DEF &MAX = MAX('NUMBER', (2,'3',67,'67',1)), DISPLAY &MAX>
```

produces:

1 67
2 3
3 4

```
<DEF &MIN = MIN('NUMBER', (2,'3',67,'67',1)), DISPLAY &MIN>
```

produces:

1 1
2 5

Note: When comparing lengths, the result depends on DEFUNITS that is in effect at that time. The following examples demonstrate this fact.

```
<defunits=in, &min = min('length', (2in,3mm,'1'))>
```

The resulting structure &MIN is &MIN = (0.1181IN,2)

```
<defunits=mm, &min = min('length', (2in,3mm,'1'))>
```

The resulting structure &MIN is &MIN = (1MM,3)

SEPARATE breaks a string into elements of a structure based either on a string, or a number. It is particularly useful to examine the results of RESERVE_INFO and FLOAT_INFO. The following example breaks the string at each blank, so each element of the resulting structure contains one word.

```
<define &string = 'Break at the blanks' >
<define &structure = SEPARATE(' ', &string) >
```

The resulting structure contains:

```
&STRUCTURE = ('Break', 'at', 'the', 'blanks')
```

The next example causes SEPARATE to make each element of the resulting structure three characters:

```
<define &structure = SEPARATE( 3, 'Break at the blanks' )>
```

makes the resulting structure:

```
&STRUCTURE = ('Bre', 'ak ', 'at ', 'the', ' bl', 'ank', 's ')
```

SEPARATE scans the string to be separated. When the break character is found, the portion of the string scanned from the character after the last break character (or the start of the string) up to but not including the found break character, is moved to the next structure element. The break character is then skipped over in the string to be separated, and scanning continues. The next portion of the string is moved to the next element in the structure.

Operators

TEXTFORM has five binary operators (they take two arguments, one on each side of the operator), and four unary operators (they take one argument, which follows the operator). There may be any number of blanks on either side of any operator. Unary operators are evaluated before binary operators.

Unary Operators

Minus Operator –

When the minus sign precedes a number the number becomes negative.

```
<DEF &NEG = –3 >
```

Since the minus operator is both unary and binary, there are several places in the TEXTFORM command language where this can lead to confusion if used incorrectly or when it is not necessary. For example, the command:

```
<INDENT BOTH 3 -2>
```

produces:

```
<INDENT  BOTH  1>
```

Plus Operator +

The plus character + is seldom used, since all numbers are positive unless otherwise stated. Like the minus operator, the plus operator is both unary and binary, so its use in the following command is unnecessary, and produces incorrect results:

```
<INDENT  BOTH  +3  +2>
```

produces:

```
<INDENT  BOTH  5>
```

Pointer Operator @

The @ is the pointer operator. It points to a name. Pointers may only be used in assignment operations, and never in expressions. Pointers may never be subscripted or substringed. Pointers are entered as an @ followed by the name you wish to point to (e.g., @&B is a pointer to &B). The validity of a pointer is only checked when it is used. The name pointed to need not be defined until you actually point to it. If you say

```
<&A  =  @&B>
```

this defines a pointer. If you say

```
<&A  =  10>
```

this sets &B to 10.

The only way to un-point a pointer is with the ATTRIBUTE command

```
<ATTRIBUTE  &A  STRING>
```

would change the pointer (@&B) in &A to a string '&B'. Note that if you make &B a pointer to &A, and &A a pointer to &B, this creates an endless pointer loop. TEXTFORM will detect this error when you attempt to use either &A or &B. with: *Endless pointer chain detected. 'name' assumed 'not defined'.*

Execute Operator $

The $ is the **execute** operator. It must be followed by the name of a variable. The contents of the variable are converted to a string (if necessary) and treated as *command* input. The variable name may be subscripted or substringed. Executed variables may appear anywhere in command mode. The executed text will be included in the source listing if the LIST EVALUATION command has been given.

```
<&A  =  'PAGE'>
<$&A>
```

would cause the PAGE command to be executed. In the next example, if your source file has strings which must be used as lengths, you can execute the string to produce a length when required. To make the parameter to macro &CHECK('4in') be a

length, define &CHECK to be:

```
<define macro &check, –
   local &var = par(1), –
   if remaining(2) < $&var, then, . . .
```

When a command takes a ‘keyword’ (this is indicated in the list of command prototypes later in the manual) the keyword may be stored in a variable and then executed.† The LINEEND command can be given a keyword to specify the type of alignment, as in:

```
<LINEEND CENTRE>
```

or

```
<DEFINE &CEN = 'CENTRE' >
<LINEEND $&CEN>
```

If THISPAGE contains the desired alignment, it too must be executed since it is a string:

```
<LINEEND $THISPAGE>
```

The execute operator lets you define names based on the contents of other variables, rather than actually typing in the name being defined. This technique is used in the FLOAT example on page 62. (The : character used in this example is described on page 111):

```
<DEFINE &A = '&CTR', DEFINE &B = 1>
<DEFINE &EX = &A:&B>
<DEFINE $&EX>
<&B = &B + 1>
<DEFINE $&EX>
```

defines variables &CTR1 and &CTR2.

If the execute operator is used in a macro, the *contents* of the executed name are stored in the macro, rather than the name itself. This produces a warning message, although it may be what you want. For example, you may need a macro which always returns to the *current* value of linespace, although you do not presently know that value. In the macro, include:

```
<LINESPACE = $LINESPACE>
```

If LINESPACE is 4MM, this will store LINESPACE = 4MM in the macro.

In many cases, you will not want the contents of the executed item stored in the macro. There are two ways to prevent this. If you do not want a LINEEND LEFT command to be stored in the following macro:

```
<define macro &top, –
```

----

†If the command takes an ‘expression’ it is not necessary to execute it, as described on page 32.

```
  pnctr, lineend $thispage, –
edef macro &top>
```

give the command containing the executed name as a string. The contents of the string are not evaluated until the macro (and therefore the string) is included in the input:

```
<define macro &top, –
  pnctr, '<lineend $thispage>', –
edef macro &top>
```

Alternately, if the name is not defined, its contents will not be stored in the macro, although a warning is produced.

```
<DEFINE &FLCTR=1>
<DEFINE MACRO &FLFIG, –
   DEF $&FL='<VS 3IN>', –
   . . . . . .
<ENDDEFINE MACRO &FLFIG>

<DEF &FL=('&', @&FLCTR)>

<&FLFIG>
```

produces the warning *&FL is not yet defined. Execution of &FL deferred until macro is used.*

If the definition of &FL appears before &FLFIG is defined, it produces the warning *Contents of &FL included in macro definition.*

Binary Operators

TEXTFORM will convert the types of the binary arguments to match the type required by the operator. Not all conversions can be made however, and in this case TEXTFORM gives an error, and returns the first argument as the result. Here are the five binary operators, examples of their use, and type of result, depending on the arguments. Note that the + and – operators can be unary or binary.

Addition Operator + and Subtraction Operator –

addition or subtraction	length	scaled	number	hex
length	length	error	error	error
scaled	error	scaled	scaled	scaled
number	error	scaled	number	number
hex	error	scaled	number	hex

Operations with one argument of type length, and the other argument a zero are allowed.

```
<&a = 5> <&b = 3>
<&a+&b> is 8
<10+&b> is 13
<2-&b> is –1
```

Variables

*Multiplication Operator **

multiplication	length	scaled	number	hex
length	error	length	length	length
scaled	length	scaled	scaled	scaled
number	length	scaled	number	number
hex	length	scaled	number	hex

```
<&a = 5> <&b = 3>
<&a*&b> is 15
<10*&b> is 30
```

Division Operator /

Where two types of result appear in the table (scaled and number), the result will only be converted to number if all four decimal places are zero.

division	length	scaled	number	hex
length	scaled number	length	length	length
scaled	error	scaled number	scaled number	scaled number
number	error	scaled number	scaled number	scaled number
hex	error	scaled number	scaled number	hex

```
<&a = 5> <&b = 3>
<&a/&b> is 1.6666
<10/&a> is 2
```

There is another way to do integer division, by using DIV, which treats two numbers as integers (rounding if necessary) and divides:

```
<DIV(5,3)>
```

produces 1. REM performs integer division, but produces the remainder from the division.

```
<REM(13.7,5)>
```

produces 4.

Catenation Operator :

Catenation joins two items into one string. The two items are converted to strings, if necessary (there can be no error).

```
<&A:14> is '514'
<'some words':' more words'> is 'some words more words'
```

Catenation can be used to force TEXTFORM to immediately evaluate text entered in command mode. (The execute operator, described earlier, causes TEXTFORM to evaluate commands immediately.) This is used in table of contents examples on

page 130.

Expressions

Expressions are items in command mode that must be evaluated by TEXTFORM, such as 1+2. They are evaluated from left to right. There is no operator priority (except for the unary operator execute, $, which is evaluated immediately). If parts of an expression are enclosed in parentheses, they are evaluated first. If a parenthesis is the first thing encountered in the expression, it makes the expression into a structure.

<&a = 3>
<&b = 5>
<&c = 10>

<&A+4-10:'22'+9> is -313
<&A:&b:&C> is 3510
<2+(4*&A)> is 14
<1in*1in/1in> produces the error *Both arguments of multiplication have type LENGTH.*

If ‘expression’ appears, the expression is evaluated, and its result is displayed in the text. Any variables used in the calculation will have their DISPLAY attributes (see page 123) applied before the expression is done.

MACROS

Macros are collections or commands and/or text which can be stored under a user-defined name. A macro is created with a DEFINE MACRO command, and terminated with an ENDDEFINE MACRO command. The macro is identified by a name following the same rules as variable names.

```
<DEFINE MACRO &M1>
<I B 3>
Macros can contain much more text than variables.
<F 2>You may switch between text and command mode
as often as you wish. The macro is terminated by
an ENDDEFINE MACRO command: don't forget it!
<F, I B 0, ENDDEFINE MACRO &M1>
```

By placing <&M1> in the subsequent text you produce:

> Macros can contain much more text than variables. *You may switch between text and command mode as often as you wish. The macro is terminated by an ENDDEFINE MACRO command: don't forget it!*

Any valid TEXTFORM input can appear between:

> <**DEF**INE **MA**CRO name>
> <END**DEF** **MA**CRO name>

However, before you use macros, remember that there are a few things which can cause errors that are difficult to find:

- starting but not ending definition of a second macro inside the first macro. This causes an error when the first macro is used.
- forgetting the ENDIF for an IF command (described on page 156) inside a macro. This causes an error at the end of the run.

Modifying the Action of the Macro When it is Used

In the previous example, the action of the macro could not be modified. Once defined, the macro did the same thing each time it was used. Macros become more versatile with the addition of **parameters**, which allow you to replace or supply information when you use the macro. Parameters are referenced through the name PAR, and must be referred to by number. That is, the first parameter to *each* macro is PAR(1), the eighth is PAR(8). PAR or PAR(0) refers to *all* the parameters in a macro. Up to 65 parameters are allowed.

The following macro accepts one parameter. It centres it, in FONT 2, at the top of a page.

```
<DEFINE MACRO &CHAP, –
    PAGE, FONT 2, PAR(1), –
    FONT 1, LINEEND CENTRE, NEWPARA, –
ENDDEFINE MACRO &CHAP>
```

To supply parameters to a macro, give the name of the macro, followed by the parameters within parentheses. Blanks before or after the parentheses are optional. If the parameter is a string of text, as a chapter title might be, enclose the text within delimiters:

```
<&CHAP( 'In The Beginning' )>
```

If the parameter is in a variable:

```
<DEFINE &BEG = 'In The Beginning' >
<&CHAP( &BEG ) >
```

TEXTFORM replaces the PAR(1) part of the macro with the parameter, and on a new page prints:

In The Beginning

If you use the macro with a different parameter, TEXTFORM will carry out the same instructions but insert the new parameter. Parameters may be strings, numbers, variables, etc.

The following macro accepts two parameters:

```
<DEFINE MACRO &TA, –
    LEND, VS 4MM, FONT 3, –
    PAR(1), LEND, FONT 2, –
    PAR(2), FONT 1, NEWPARA, –
ENDDEFINE MACRO &TA>
```

When you use the macro, separate the parameters with a command separator:

```
<&TA('Introduction' , 'D. Johnston')>
```

produces:

Introduction
D. Johnston

To omit one of the parameters, leave that position between the commas blank. This is the same as the null parameter ''. To check if a parameter has been omitted, the test is:

```
<IF PAR(1) = '', . . .
```

TEXTFORM carries out all the commands within a macro, even if some parameters are missing. If there are no parameters, the parenthesized list need not appear.

In the following examples, the first parameter is null:

```
<&TA( , 'D. Johnston')>      <&TA( '', 'D. Johnston' )>
```

produces:

D. Johnston

If you make errors in entering the parameters (for example, forget to enter a delimiter) *several* errors messages are produced, including *String delimiter missed*, *Bad parameter list format*, and *Unmatched parentheses*.

The continuation character can be used between parameters to enter them on two lines:

```
<&TA('Introduction', –
'D. Johnston')>
```

If a single parameter is too long to enter on one line, it can be split as follows:

```
<&TA(' A long parameter may be ': –
'entered on several lines')>
```

You *cannot* do the following:

```
<&TA(' A long parameter may not be –
entered on several lines like this')>
```

A long parameter may also be entered as a variable:

```
<DEFINE &HDNG = ' This is the long heading ' >
<&TA( &HDNG )>
```

The same effect would be produced with:

```
<&TA( '<&HDNG>' )>
```

If the parameter contains a footnote reference, as in

Written by Jones and White[1]

do not enter the entire footnote in the parameter if a table of contents entry is being made by the macro. If you do this, the footnote commands and text will also be stored in the table of contents file, and will reappear when the contents are generated. Instead, do the following:

```
<DEFINE &F1='<FOOT>This is the footnote.<EFOOT>'>
<&HEAD('Written by Jones and White<&F1>')>
<&F1=' '>
```

so that &F1 is empty when the contents are generated. If the text of the footnote is too long for a variable, it must be stored in a macro, which should be erased and redefined as an empty macro or variable after the heading macro.

Changing the Macro

To change a macro permanently, it must be erased and then defined again. The format of the ERASE command is:

```
<ERASE name>
```

Text Within Macros

Each line ended with a command terminator, or without a continuation character, indicates a word ending, where a word space will be inserted. If you define the macro:

```
<DEFINE MACRO &EA>
<FONT 2>et¬al<FONT 1>
<ENDDEFINE MACRO &EA>
```

there will always be a word space after the word ‘al’. To prevent this, stay within command mode inside the macro, and include the text as a string:

```
<DEFINE MACRO &EA, –
FONT 2, –
'et¬al', –
FONT 1, –
ENDDEFINE MACRO &EA>
```

or:

```
<DEFINE MACRO &EA, –
FONT 2>et¬al<FONT 1, –
ENDDEFINE MACRO &EA>
```

Sample Form Letter Macro

The following example shows a sample form letter in which only the name and address, and several details are changed. Every use of the &LETTER macro prints a complete letter.

```
<DEFINE MACRO &LETTER>
  <PAGE, VERTSPACE 1IN, DATE>
  <MONTH, ' ', YEAR>
  <LINE> P.D.Q. Parts Ltd.
  <LINE> Anywhere, Allstate.

  <LINEEND, VERTSPACE 8MM, PAR(1)>
  <LINE, PAR(2)>
  <LINE, PAR(3)>
  <LINE, PAR(4)>

  <LINE, VERTSPACE 8MM> Dear <PAR(1)>:
```

```
  <NEWPARA> This is to advise you that your <PAR(5)>, which you
ordered in
  <PAR(6)> of <PAR(7)>, has arrived and may be collected
  at the above address. Thank you for your
  <IF PAR(7) ¬= YEAR, THEN, 'extreme', ENDIF> patience.

  <LINEEND, VERTSPACE LINESPACE>
  Sincerely,
  <LINEEND, VERTSPACE .5IN> J. Smith
  <LINE> Parts Department
<ENDDEFINE MACRO &LETTER>

<&LETTER('Mrs. Jones', '123 4th Ave', 'Your town', –
'Your province', 'widget', 'September', '1966')>
```

20 January 2010
P.D.Q. Parts Ltd.
Anywhere, Allstate.

Mrs. Jones
123 4th Ave
Your town
Your province

Dear Mrs. Jones:

This is to advise you that your widget, which you ordered in September of 1966, has arrived and may be collected at the above address. Thank you for your extreme patience.

Sincerely,

J. Smith
Parts Department

More Control over Macros

You can have more control over the macro when you check the type or number of the parameters supplied to a macro. You can also control where the text produced by the macro appears. This is useful to prevent widows (headings standing alone at the bottom of a column).

For example, you may write a macro to print a heading *and* test the amount of space remaining on a page. If the macro is not used with two parameters (the system variable NRPARS contains the number of parameters to a macro), an error message

is produced:

```
<DEFINE MACRO &TEST>
  <IF NRPARS ¬= 2, THEN, –
    ERROR 4 '&TEST parameters must be heading, length', ENDIF>
  <LINEEND, VERTSPACE 2*LINESPACE>
  <IF REMAINING(2) < PAR(2), THEN, PAGEEND, ENDIF>
  <FONT 3, PAR(1), FONT 1, NEWPARA>
<ENDDEFINE MACRO &TA>
```

The second parameter supplied to this macro must be a length, as in

```
<&TEST('Heading', 3IN)>
```

It indicates how much space must be remaining on the page before the heading can be printed. A simpler test is:

```
<IF REMAINING(2) < 3IN, THEN, PAGEEND, ENDIF>
```

Inside a macro, TEXTFORM cannot evaluate &A=PAR(n)(1;3). Put PAR(n) into a local variable before trying to substring it.

TYPE and VTYPE can be used to check parameters to a macro. STACK and UNSTACK save and restore the values of variables, and are often used within macros.

```
<DEFINE MACRO &HEAD, –
  STACK(ALIGNMENT), ALIGNMENT=CENTRE, –
  PAR(1),LINEEEND, UNSTACK(ALIGNMENT), –
  NEWPARA, –
ENDDEFINE MACRO &HEAD>
```

You can only STACK names of variables which you are allowed to change. A system variable, such as CURFONT, which TEXTFORM changes, cannot be stacked. In this case, do the following:

```
<DEFINE &SAVEFONT>
<DEFINE MACRO &HEAD, –
  INDENT BOTH 0, &SAVEFONT = CURFONT, –
  PAR(1), FONT &SAVEFONT, –
ENDDEFINE MACRO &HEAD>
```

When input is being generated from a macro, the system variable MACFLAG is set to TRUE. It is FALSE when source lines are being processed.

```
<DEFINE MACRO &TEST,-
'The value of MACFLAG is ',MACFLAG,-
ENDDEFINE MACRO &TEST>
<&TEST>
```

produces:

```
The value of MACFLAG is #01#
```

Details about the Macro Definition and Use

When a macro is defined, its name and contents (in their original form), are saved in the symbol table. After the DEFINE MACRO . . . line, you may give any valid TEXTFORM input, and you may either stay in command mode, or return to text mode. If a macro is defined within a macro, parameter replacement (from the outer macro), will not occur in the internally defined macro. Executed variables which appear in the definition will generate a warning, whether or not they are defined. Regardless of whether the executed variable is defined or not, the macro being defined will be executable. If you do not want the variable executed during the macro definition, it must not be defined at the time of the definition. If it is defined at the time of the macro definition, its contents will replace its reference in the macro definition.

If you wish one of the parameters to the outer macro to appear as part of the definition of the inner macro, you must use an executed variable.

```
<DEFINE MACRO &X>
. . .
<LOCAL &A = PAR(1)>
. . .
<DEFINE MACRO &Y>
. . .
<$&A>
. . .
<ENDDEFINE MACRO &Y>
. . .
<ENDDEFINE MACRO &X>
```

When a macro call is recognized, TEXTFORM saves the parameters, then suspends the input line at the end of the parameter list. If LIST EXPANSION has been specified:

- the macro name, and nesting level, are printed in the source listing.
- when execution of the macro ends, the macro name and nesting level are again printed in the source listing.
- if TEXTFORM is returning to another macro at this point, its name is printed in the source listing.

TEXTFORM now queues the body of the macro for input. A **queue** is a list of items to be processed; in this case, the contents of the macro. The body of the macro appears in the input exactly as it was defined; the only exception being that wherever a PAR(expression) appears, it is replaced with the exact parameter which appeared in the macro call. If parameter ‘expression’ did not appear a null string will be used. If the LIST EXPANSION command has been given, the macro text (as defined) will appear in the source listing. (There will be no indication of parameter replacement in the source listing unless LIST PARTRACE has been specified). Note that the current INPUTMODE, and input translation, will be applied to this queued input. Here are some errors which are peculiar to PAR(expression): If parameter, such as parameter 4, has been omitted, assignment to it will cause an error. For example, if PAR(4) was omitted,

```
<PAR(4) = 116>
```

is the same as:

```
<'' = 116>
```

which is an error.

If the type of the parameter is improper, errors may be generated. If PAR(3) is 'some words'

```
<PAR(3) = PAR(3)+1>
```

is the same as

```
<'some words' = 'some words'+1>
```

which has several errors in it.

Storing a Macro in a Separate File

There is a method in TEXTFORM to let you store the contents of a macro, or variable, in a separate file. See page 162. This might be useful, for example, if you want to save certain macros for a separate TEXTFORM run.

Recursion

A macro which calls itself is a **recursive** macro. Macros may call themselves, either directly or indirectly. Recursion must be used with conditional control (discussed on page 156), otherwise there is no way to terminate the recursion. This would eventually result in macro storage overflow. In general, if a FOR or WHILE command can be used instead of recursion, it is cheaper.

The following macro prints par(1) until its value reaches par(2):

```
<DEFINE MACRO &N, –
   PAR(1)>
   <IF PAR(1)<PAR(2) THEN &N(PAR(1)+1, PAR(2)), –
   ENDIF, –
ENDDEFINE MACRO &N>

<&N(1,3)>
```

produces:

```
1 2 3
```

Local Variables

The LOCAL command is used to define a variable inside a macro. The variable can only be used inside the macro, and disappears when execution of the macro ends. This is in contrast to a **global** variable, which can be used anywhere in a TEXTFORM run, once it has been defined. Note that the more LOCAL variables that you use in a macro, the slower TEXTFORM will run in command mode, so only use

LOCAL variables when a global variable cannot be used.

The command to define a local variable is:

<**LOC**AL name>

'name' is the name to define. It has the same characteristics as a variable. If no value is specified, a null string is used.

<LOCAL &QUICK>

An initial value can be assigned when the variable is defined:

<LOCAL &QUICK = 1>

INFORMATION ABOUT USER-DEFINED NAMES

Erasing Names

User-defined names can be erased with the command:

```
<ERASE  name>
```

When this command is given, the name becomes un-defined and can no longer be referenced. You can redefine an item with the same name as one which has been erased.

Checking Whether Name Exists

You can use EXIST to tell you whether a name has been defined. Its result is TRUE or FALSE.

```
<IF  EXIST(&NAME)  =  FALSE,  THEN,  –
    DEFINE  &NAME  ,  –
ENDIF>
```

Checking Type of a Name

Using TYPE tells you the type of a name as one of:

VARIABLE
MACRO
FUNCTION
COMMAND
PAGE (for both logical page and table definitions)

For example:

```
<TYPE(LINESPACE)>
```

produces:

```
VARIABLE
```

Giving Attributes to a Name

The ATTRIBUTE command changes or assigns attributes of user-defined names, variables, commands, macros, or structure elements. Not all of these items can be given all of the attributes. Some are allowed only for variables, while others are only for commands. The format of the command is:

<**ATT**RIBUTE name kwd . . . >

'name' can have more than one attribute given to it in the same command. Each 'kwd' represents one attribute. The complete list of attributes is Appendix 1. If you want a value such as &A to remain **constant**, for example, so that it can never be changed, give it this attribute:

<ATTRIBUTE &A CONSTANT>

Any subsequent attempt to change &A will produce the error *Attempt to change a constant. The only 'change' allowed is erasure.*

Some attributes are only allowed for variables; others may be used with commands, macros, or structure elements. If a command name is given, the attribute(s) given apply only to that form of the command (for example, an attribute for NP does not apply to NEWPARA). Here is the list of attributes:

kwd

The following attributes are valid for all items:

AXR

Every reference to 'name' will be cross referenced, regardless of the cross reference settings.

NXR

'name' will never be cross referenced.

RESTRICTED

Changes to 'name' (if it is a variable), or use of 'name' (if it is anything else), will only be allowed from predefined (i.e. TEXTFORM's) macros. Any other attempts will result in the error: *Assignment to this item is RESTRICTED.*

The following attributes are valid for variables:

CONSTANT

The value of 'name' may not be changed in future, but may be ERASEd if it is user-defined.

DISPLAY type

DISPLAY controls the way in which variables are printed in the text. ‘type’ may be any of the following (but only up to one from each group):

type (valid only for integer numbers or strings which can be converted to integer numbers):
ARABIC – digits
ENGLISH – words
FRENCH – words
ROMAN – roman numerals
ALPHABETIC – letters (a-z, aa-zz etc.)

which may be in (valid for all variables):
UPPER**C**ASE – upper case
LOWER**C**ASE – lower case

```
<DEFINE &YEAR = YEAR>
<ATTRIBUTE &YEAR DISPLAY ROMAN LC>
<&YEAR>
```

produces:

```
mmx
```

INCR = string3

Increment ‘name’ by ‘string3’ after display. This may not appear with ‘LIST’.

```
<DEFINE &CTR = 1>
<ATTRIBUTE &CTR INCR = '1'>
<&CTR, NEWLINE, &CTR, NEWLINE, &CTR>
```

produces:

```
1
2
3
```

LIKE name2

assume attributes of ‘name2’. Attributes which follow LIKE override the attributes copied from name2. The current value of ‘name’ is not checked for validity.

LIST = structure†

‘structure’ contains allowable values. The current value of ‘name’ is not checked for validity. ‘LIST’ may not appear with ‘INCR’.

```
<DEFINE &A = 1, ATTRIBUTE &A LIST = (1,2,3,4,5)>
```

----

† If any of MAX, MIN, or LIST are specified, all assignments containing this variable name will be checked, and an error produced if the conditions are not met.

```
<&A = 9>
The value of &A is: <&A>
```

produces:

Value not found in values list.
The value of &A is: 1

MAX = string4

maximum value that may be assigned. The current value of ‘name’ is not checked for validity.

```
<DEFINE &A = 5>
<ATTRIBUTE &A MAX = '8'>
<&A = 10, &A>
```

produces:

Value higher than maximum allowed. Assignment not performed.
5

MIN = string5

minimum value that may be assigned. The current value of ‘name’ is not checked for validity.

```
<DEFINE &B = 5>
<ATTRIBUTE &B MIN = '2'>
<&B = 1, &B>
```

produces:

Value lower than minimum allowed. Assignment not performed.
5

STRING

Not fully implemented – except POINTER to STRING. ‘name’ will be changed to a string.

UPPER**C**ASE

All values assigned to ‘name’ will be converted to upper case before being saved. ‘name’ is also given the FIXED attribute. The current value of ‘name’ is not checked for validity. The contents of ‘name’ must be a STRING when the attribute is specified, and the string must be assigned to itself for the attribute to take effect on the current string.

```
<DEFINE &STR = 'title'>
<ATTRIBUTE &STR UPPERCASE, &STR = &STR>
<&STR, &STR = 'New title', LINE, &STR>
```

produces:

```
TITLE
NEW  TITLE
```

LOWER**C**ASE

All values assigned to ‘name’ will be converted to lower case before being saved. ‘name’ is also given the FIXED attribute. The current value of ‘name’ is not checked for validity. The contents of ‘name’ must be a STRING when the attribute is specified, and the string must be assigned to itself for the attribute to take effect on the current string.

STATIC

‘name’ must remain the same length or shorter. Using this feature, strings can be forced to have fixed length.

```
<DEF &LEN = 'aaaa', ATTRIBUTE &LEN STATIC>
<&LEN = 'xxxxxxxxx', &LEN>
```

produces:

Item assigned is STATIC. Result of expression is too long. Result truncated.
xxxx

DYNAMIC

‘name’ will be allowed to move in storage. (Opposite of STATIC.)

FIXED

The type of ‘name’ (e.g. STRING, LENGTH, etc.) will be fixed. All future assignments to ‘name’ will be converted to its present type (if possible) before assignment.

```
<&F = 1,ATTRIBUTE &F FIXED>
<&F = 'a'>
<&F>
```

produces:

Number too large to scale. Truncated to integer portion only.
0

VARTYPE

The type of ‘name’ will be freed. (Opposite of FIXED.)

The following attributes are valid for structures:

FIXEDELTYPE

All assignments to elements of the structure will be forced to have the same type as the first element of the structure.

```
<DEFINE &STRUC = (1)>
<ATTRIBUTE &STRUC FIXEDELTYPE>
<&STRUC = (1, 'a')>
<DISPLAY &STRUC>
```

produces:

Number too large to scale. Truncated to integer portion only.
1 1
2 0
3

FIXEDNRELS

Any assignment to the structure which would cause the number of elements in the structure to change will be flagged as an error.

```
<DEFINE &STRUC = (1,2,3)>
<ATTRIBUTE &STRUC FIXEDNRELS>
<&STRUC(4) = 5>
```

produces:

Result of expression cannot be converted to FIXED type requested. Assignment not performed.

Information About the Attributes of a Name

ITYPE produces a structure that contains an interpretation of all the attributes of a name. For structures, if an index is provided after the name of the structure, the attributes for that specific structure element are returned. For example:

```
<&x = ITYPE(linespace), display &x>
```

produces all the attributes for LINESPACE:

1 PREDEFINED
2 VARIABLE
3 NUMBER
4 NUCLEUS
5 STATIC
6 FIXED TYPE
7 LENGTH
8 AT ASSIGN

while:

```
<&X = ITYPE(leftindents, 2), display &x>
```

produces the attributes for the second element of LEFTINDENTS:

1 VARIABLE
2 NUMBER
3 LENGTH

TABLE OF CONTENTS AND INDEX

These two items may seem unrelated, but TEXTFORM handles them in the same way. In both cases, you provide the entry and the page number, which TEXTFORM stores in a separate file. You ask for the table of contents or index to be printed once all the information has been collected. This is usually at the end of the TEXTFORM run, so you must then move the table of contents pages to their proper location in the front pages of the document.

Table of Contents

Here is a sample which generates a table of contents. The following discussion will refer to this sample, and show how to change it. Notice that the page number of this manual, 129, is used in this example.

```
<DEF  &SB  =  'Subgenus'>
<TOC(0,  PNCTR,  'Genus')>
<TOC(1,  106,  &SB)>

<TOC(0)>
```

produces, on a new page:

Table of Contents

Genus 129
    Subgenus 106

Entries to Table of Contents

Each entry made using TOC indicates:

1. the level of importance in the table of contents, which is usually indicated by indenting
2. the page number of the entry, which will appear in the contents
3. the heading itself, which will appear in the contents.

The command to store an entry in the contents has the format:

<TOC(level [, page, $entry_1$ [, . . ., $entry_8$]])>

level
is the level of the entry. Levels are 0 to 5, and are usually indicated by indents (see the following instructions on modifying the format of the table of contents to change this). The table of contents is printed when ‘level’ is negative, or the *only* parameter. A main level entry in contents is entered as:

```
<TOC(  0,  . . .
```

The command to print the table of contents is:

```
<TOC( 0 )>
```

page
is the page number of the entry. You can insert the page number manually, as in:

```
<TOC( 1, 66 . . .
```

or you can use PNCTR. This causes TEXTFORM to store the value of the *current* page number, and is the recommended practice. It permits you to add or delete pages from your document, and let TEXTFORM make the changes in the table of contents.

$entry_1$ through $entry_8$
are the entries, often headings, for the contents. They may contain parameters, strings, variables, or other commands. In most instances, only '$entry_1$' is used. Up to eight entries can be given, however, if you need to modify the format of the contents. An example of this appears on page 134.

If you enter a string, enclose it in delimiters:

```
<TOC( 0, PNCTR, 'Genus' )>
```

If you enter a variable name, the contents of the variable are stored in the contents:

```
<TOC( 1, 66, &SB )>
```

However, if the entry is:

```
<TOC( 1, 66, '<&SB>' )>
```

the *name* of the variable is stored in the contents. If the value of &SB is changed before the contents are generated, the *new* value of &SB will appear, not the value which was in effect when the entry was made.

You may want to italicize a variable when it appears in the contents. If &A contains 'Subgenus', the following will *not* work, because it will not evaluate &A until the table of contents is produced:

```
<TOC(0, PNCTR, '<FONT 2, &A, FONT 1>' )>
```

The above would store '<FONT 2, &A, FONT 1>', and &A might be changed before the table of contents. Instead, catenate &A to the FONT command to force TEXTFORM to evaluate &A immediately:

```
<TOC(0, PNCTR, '<FONT 2>':&A:'<FONT 1>' )>
```

This stores the following for the table of contents: '<FONT 2>Subgenus<FONT 1>'.

If the entry is being made inside a macro, use the parameter to the macro, PAR(1) in the next example, to supply the parameter to TOC. When doing this, don't forget the closing parenthesis:

```
<DEFINE MACRO &HEAD>
```

```
<TOC( 0, PNCTR, PAR(1) )>
<ENDDEFINE MACRO &HEAD>

<&HEAD( 'Subgenus' )>
```

If &CT is a variable containing a counter:

```
<DEFINE &CT = 1>
<TOC( 0, PNCTR, &CT:'. ':PAR(1) )>
```

produces:

```
1. Subgenus .......................................... 131
```

If &CT has a display attribute (see the ATTRIBUTE command) it must be displayed in the TOC entry using FORDIS. Otherwise, the value of &CT will appear as 1 in the contents.

```
<DEFINE &CT = 1>
<ATTRIBUTE &CT DISPLAY ALPHABETIC UPPERCASE>
<TOC( 0, PNCTR, FORDIS(&CT):'. ':PAR(1) )>
```

produces:

```
A. Subgenus .......................................... 131
```

FORDIS converts its parameter as though it were to be displayed. Any INCREMENT (see ATTRIBUTE) for the parameter is also done. The parameter cannot be substringed or subscripted.

TEXTFORM provides other alternatives to change how a number is displayed. If you want the page number to be a roman number in the table of contents, use:

```
<TOC( 0, ROMAN(PNCTR), PAR(1) )>
```

which produces:

```
Subgenus ......................................... cxxxi
```

If you want an upper case English page number, use:

```
<TOC( 0, UPPERCASE(ENGLISH(PNCTR)), PAR(1) )>
```

which provides:

```
Subgenus ....................... ONE HUNDRED THIRTY-ONE
```

Printing the Table of Contents

You can generate a table of contents at any point in a TEXTFORM run, by making a TOC entry with a negative 'level', or with only one parameter. When you do this, TEXTFORM inserts a variable called TOCHEADER in the input to produce the title, and then formats the contents. As shown in the next section, TOCHEADER begins a

page, sets its own indents, and prints ‘Table of Contents’. If you then continue to make more TOC entries, the next table of contents will contain only what has been entered since the last contents. New indents will also be in effect, unless you reset them.

The table of contents can be printed automatically using the AT ENDOFFILE command. The following lines, which would be inserted near the beginning of a file, tell TEXTFORM to include the name &PRINTOC in the input at the end of the file. &PRINTOC then generates the contents.

```
<DEFINE &PRINTOC = '<TOC(0)>'>
<AT ENDOFFILE &PRINTOC>
```

If you are producing page numbers with a reserve command, &PRINTOC could be modified to end the page, suspend any reserves, begin page numbers at iii at the bottom centre of the page, and then generate the table of contents.

```
<define &bot = '<vs .5in, roman(pnctr), lend c>' >

<define macro &printoc>
  <pageend, suspend reserve top, suspend reserve bottom>
  <topmargin=1in, botmargin=0in>
  <reserve bottom page size default 1in &bot>
  <pnctr = 3>
  <toc(0)>
<edef macro &printoc>
```

How the Contents are Stored

When you make the first TOC entry, TEXTFORM checks to see if a file called –TXTFTC01 exists. If it does, TEXTFORM empties it, otherwise it creates the file, with a size of 10 pages. If you create the file, TEXTFORM does not destroy it after the table of contents is generated. In this way you can keep a copy of the table of contents entries.

If you want to ensure that the file is saved, create it as part of the TEXTFORM run before the first TOC entry:

```
<MTSCMD('$create –txtftc01')>
```

The name of the file created is determined by the variable TOCINDEX, discussed in the next section. By default, TOCINDEX=1 and the filename is –TXTFTC01.

Lines in these files contain the information you provided with the TOC entry, and are stored as:

```
<$TOCMACROLIST(TOCINDEX)( level, –
"page", –
"entry1", "entry2", . . . )>
```

If necessary, you can check the file before the table of contents is produced if your file contains:

```
<DEFINE &PRINTOC = '<TOC(0)>'>
```

```
<AT ENDOFFILE &PRINTOC>
<AT ENDOFFILE MTS>
```

Enter the command to run TEXTFORM, and wait for it to return to MTS. Then check the file, and restart.

```
# run *textform scards=file . . .
# edit –txtftc01
:comment check the file
:comment do not change the format of macro calls
:stop
# restart
#
```

Note: this file can be edited, but the form of the macro calls in it must remain as they are created (ending with >) or there may be errors when the table of contents is processed.

At the end of the TEXTFORM run, the following warning is printed if the table of contents has not been generated: *Table of Contents number* nn *contains text but has not been produced.*

Modifying the Format of the Table of Contents

Before the table of contents is printed, the variable TOCHEADER is placed in the input by TEXTFORM. TOCHEADER is equivalent to the following:

```
'<PEND, ALIGNMENT=LEFT, I B 0, F 3> Table of Contents':–
'<F 1, LEND C, VG LINESPACE>':–
'<LEFTINDENTS=(.2IN,.4IN,.6IN,.8IN,1IN,1.2IN)>':–
'<RIGHTINDENTS=(.5IN), SPLITSTRING = "." >'
```

By changing this variable, you can adjust the title which appears before the contents. For example:

```
<TOCHEADER = '<LEND, VS LINESPACE, I B OFF, F
2>Contents<F, LEND>'>
<TOC(0)>
```

The above example spaces down, turns off indents, and prints a title *Contents* before generating the table of contents. Indents are not changed.

When the TOC(0) command is given, the stored table of contents entries are passed, one at a time, to a predefined macro called TOCMACRO, which formats the contents. This macro contains:

```
LEND,I B PAR(1) 1, I H=1 L PAR(1)+1,KE,–
PAR(3),PAR(4),PAR(5),PAR(6),PAR(7),PAR(8),PAR(9),PAR(10),–
" ",I R OFF,SP,PAR(2),EKE,LEND,–
```

Multiple Tables of Content

Multiple tables of content can be produced by changing the variable TOCINDEX. By default, TOCINDEX=1 and all contents entries are stored in the file –TXTFTC01. TOCINDEX can have a value from 1 to 99. If it is out of this range, no contents entries are made. When TOCINDEX is within the range, entries are stored in a file called –TXTFTCnn, where 'nn' is the value of TOCINDEX.

When the TOC(0) command is given, TEXTFORM checks a list of TOCHEADER names stored in TOCHEADERLIST, and uses the name in position TOCINDEX of this list. By default, TOCHEADERLIST contains:

```
('TOCHEADER', 'TOCHEADER', . . ., 'TOCHEADER')
```

To format the contents, TEXTFORM checks a list of macro names stored in TOCMACROLIST, and uses the macro name in position TOCINDEX of this list. By default, TOCMACROLIST contains:

```
('TOCMACRO', 'TOCMACRO', . . ., 'TOCMACRO')
```

Any of the 99 tables of content can be formatted with an alternate TOCHEADER or TOCMACRO, by changing the names in the TOCHEADERLIST and TOCMACROLIST lists.

```
<comment generate two tables of content>
<DEF &DOTOC = '<TOCINDEX=1,TOC(0),TOCINDEX=2,TOC(0)>'>
<AT EOF &DOTOC>

<comment The first table of contents uses defaults –
           but change format for second contents>
<DEF &FIGHEADER='<PEND,''List of Figures'',LEND C>':–
                      'Figure <SP '' ''> Page <VS 4MM,LEND>'>

<DEF MA &FIGMAC,–
  LEND, I B 0 1, VS 4MM, KE, PAR(3), I L PAR(1)+1, –
  PAR(4), ' ', SP, PAR(2), EKE, –
EDEF MA &FIGMAC>

<TOCHEADERLIST(2) = '&FIGHEADER'>
<TOCMACROLIST(2) = '&FIGMAC'>

<comment make entry into second table of contents>
<TOCINDEX = 2>
<TOC(0, pnctr, '1.', 'This is figure heading')>
<TOCINDEX = 1>
```

Index

The index is produced in the same way as the table of contents (see the previous section). The following sample illustrates how it is used:

```
Today let us talk about 'stuff'.
<X( 0, 15, 'stuff' )>
It will be our topic.
<X( 0, 18, 'topic' )>
There are several types of stuff, including important,
<X( 0, 23, 'stuff', 'important' )>
and, more often, irrelevant stuff.
<X( 1, 25, 'irrelevant' )>
We also talked about stuff on page 4. Put it in the index as well.
<X( 0, 4, 'stuff' )>

< X(0) >    <comment print the index>
```

produces the following index, on a new page:

Index

```
stuff,   4, 15
    important,   23
    irrelevant,   25
topic,   18
```

Entries to Index

You must make an index entry for each occurrence of a specific word that you want indexed. *The entry must be made where the word occurs in the file.* You cannot indicate a list of words that you want indexed, and have TEXTFORM automatically make the index entry whenever the word appears. The cost of doing this would be prohibitive.

Like the table of contents, each index entry has the format:

<X(level [, page, $entry_1$ [, . . ., $entry_8$]])>

level
is the level of importance of the entry. Levels can be 0 to 3, and are usually indicated by indents in the formatted index (see the following instructions on modifying the format to change this). A main entry to the index begins with:

```
<X( 0, . . .
```

The index is printed when a ‘level’ is negative, or the only parameter.

```
< X(0) >
```

page
is the page number of the reference. You can supply this yourself, or use PNCTR, or use any other information that you want sorted, such as bibliography entries, as

shown on page 139.

```
<X( 0, pnctr, . . .
```

entry$_1$ through entry$_8$
are entries and sub-entries. For example, to insert the entry ‘stuff’ into the index, use:

```
<X( 0, pnctr, 'stuff' )>
```

If any of ‘entry$_1$’ through ‘entry$_8$’ are omitted, the value of that parameter from the previous entry is copied. If you have made the index entry

```
<X( 0, pnctr, 'stuff', 'green' )>
```

then the following three entries all produce the same result in the index:

```
<X( 0, PNCTR, 'stuff', 'blue' )>
<X( 0, PNCTR, , 'blue' )>
<X( 1, PNCTR, 'blue' )>
```

If you enter a variable name, the *contents* of the variable are stored in the index:

```
<DEFINE &S= 'stuff' >
<X( 0, 66, &S )>
```

However, if the entry is a string containing the variable name:

```
<X( 0, 66, '<&S>' )>
```

the *name* of the variable is stored in the index. If the value of &S is changed before the index is generated, the *new* value of &S will appear, not the value which was in effect when the entry was made. For example, If you want the page number to be italicized, do *not* make the entry:

```
<X(0, '<F 2,PNCTR,F>', 'text')>
```

because PNCTR will not be evaluated until the index is generated. Instead, use catenation to force the current value of PNCTR to be stored in the index entry (or change XMACRO, described later):

```
<X(0, '<f 2>':pnctr:'<f>', 'text')>
```

It is convenient to use a macro to make index entries. The following example shows how to define and use the macro.

```
<DEFINE MACRO &X, –
  X(0, PNCTR, PAR(1) ), –
ENDDEFINE MACRO &X>

<&X('index entry')>
```

The next macro accepts two parameters, and enters both in the index, using each as a subentry for the other.

```
<DEFINE MACRO &X2, –
  X( 0, PNCTR, PAR(1), PAR(2) ), –
  X( 0, PNCTR, PAR(2), PAR(1) ), –
ENDDEFINE MACRO &X2>

<&X2('title', 'author')>
```

produces the following entries in the index:

```
author, title,    137
title, author,    137
```

Printing the Index

This is done in the same manner as printing a table of contents, described on page 131. The variable XHEADER, described below, is placed in the input before the index is generated.

If you want to produce both an index and a table of contents at the end of a document, it is necessary to generate the index, turn off the page numbers, and then generate the table of contents:

```
<DEFINE MACRO &PRINTALL, –
  X(0), PEND, SUS RES T, SUS RES B, –
  TOPMARGIN = 1IN, BOTMARGIN = 1IN, –
  TOC(0), –
ENDDEFINE MACRO &PRINTALL>
<AT ENDOFFILE &PRINTALL>
```

How the Index Entries are Stored

During the run, TEXTFORM saves index entries and page numbers in the temporary file –TXTFXI01, which it creates, (size=10 pages) if necessary. If you create the file, TEXTFORM will not destroy it after the index is generated. The name of the file created is determined by the variable XINDEX, discussed in the next section.

Before the index is generated, all the entries in the file –TXTFXI01 are sorted and put into the file –TXTFXS01 (which it handles like –TXTFXI01). –TXTFXS01 then produces input lines, which are formatted by XMACRO (described below).

At the end of the TEXTFORM run, the following warning is printed if index *nn* has not been generated: *Index number* nn *contains text but has not been produced.*

Modifying the Format of the Index

Before the index is printed, the variable XHEADER is placed in the input. XHEADER, which is equivalent to the following, can be modified to change the title which appears before the index is generated.

```
'<PEND,ALIGNMENT=LEFT,I B 0,F 3>Index <F 1,LEND C,VG
LINESPACE>':–
'<LEFTINDENTS=(.2IN,.4IN,.6IN,.8IN,1IN,1.2IN)>':–
```

```
'<RIGHTINDENTS=(.5IN)>'
```

Page numbers are sent to XMACRO to be formatted:

```
<XPAGENUM = ( 1, 3, 5, 9, 44, 100 )>
<$XMACROLIST(XINDEX)( 0, 6, 'pies', 'apple', )>
```

This is done automatically once a X(0) command has been given. The command $XMACROLIST(XINDEX) causes XMACRO to be done. XMACRO is:

```
LEND, I B PAR(1) 0, I H=1 L PAR(1)+1, –
XPARS=NRPARS-1, PAR(3), –
com print entries, –
     FOR 'XCOUNT=4' UNTIL 'XPARS' DO XPRINT, -
XPARS=PAR(2), –
com print page numbers, –
FOR 'XCOUNT=1' UNTIL 'XPARS' DO XPGPRINT, –

<XPRINT='<'', '', PAR(XCOUNT)>'>
<XPGPRINT='<'', '', XPAGENUM(XCOUNT)>'>
<com XPARS and XCOUNT are predefined>
```

In the macro, XPARS is given a value equal to the number of parameters minus 1 because the macro call ends with a null parameter.

Multiple Indices

XINDEX determines which index is being filled (just as TOCINDEX determines which table of contents is being filled). The list of XHEADER names to use are stored in XHEADERLIST. By default,

```
<XHEADERLIST = ('XHEADER', 'XHEADER', . . ., 'XHEADER')>
```

so that XHEADER is used for all 99 indices. Different names can be used for the various indices by changing XHEADERLIST.

The structure XMACROLIST contains the macro names used to format the index entries. By default,

```
<XMACROLIST=( 'XMACRO' , 'XMACRO', . . . , 'XMACRO' )>
```

so that XMACRO is used for all 99 indices. You can supply your own macro name, as described in the examples for multiple tables of content.

The following example shows how to combine the indices from two separate runs. When doing this, LEVELS and MAXLEN must be the same in each run. These are described on page 141.

```
RUN 1
<mtscmd('$create -txtfxi01')>
<comment so TEXTFORM does not destroy it>
<index on levels=3 maxlen=64>
<x(0,1,'AAAAA')>    <x(0,2,'BBBBB')>
<x(0)>
<mtscmd('$rename -txtfxi01 INDEX1')>
<comment this saves the unsorted index>
```

```
RUN 2
<mtscmd('$create -txtfxi01')>
<comment so TEXTFORM does not destroy>
<index on levels=3 maxlen=64>
<x(0,1,'YYYYY')>
<x(0,2,'ZZZZZ')>
<def ma &storeindex, –
    mtscmd(''$edit –txtfxi01 :i *l '$continue with INDEX1' ''), –
    x(0), mtscmd(''$rename -txtfxi01 INDEX2''), –
edef ma &storeindex>
<at eof &storeindex>
```

Handling Large Indices

During a run, TEXTFORM checks whether lines have been read from SCARDS during a specified interval of time. This interval of time is stored in TIMERLIMIT; its default is 1 CPU second. As TEXTFORM reads the index lines from –TXTFXS01 this limit may be reached; you will be asked if you wish to continue. To prevent the interrupt when you have a large index, increase the value of the variable TIMERLIMIT before generating the index. For example, to double the value of TIMERLIMIT:

```
<DEFINE &PRINTNDX = '<TIMERLIMIT = TIMERLIMIT*2, X(0)>' >
```

Using Index for Bibliography

If bibliography references are stored in macros, they can be used to produce a bibliography where names are sorted, and appear only once regardless of how many times in the document they are referenced:

```
<comment define two items for each reference>

<def &west=('West', ', Donald J.', '1973')>
<def ma &refwest>
   Who Becomes Delinquent? London: Heinemann. <edef ma &refwest>

<def &west2=('West', ', Donald J.', '1977')>
<def ma &refwest2>
   The Delinquent Way of Life. London: Heinemann.
<edef ma &refwest2>

<def &westj=('West', ', James H.', '1927')>
<def ma &refwestj>
   Who Becomes Delinquent In the Twenties? New York: Oxford.
<edef ma &refwestj>

<def &smi=('Smith', ', John Q.', '1986')>
<def ma &refsmi>
   Urban Turmoil: The Politics of Hope. New City: Polis Publishing
Co. <edef ma &refsmi>

<comment &REF puts the reference in the text –
and makes the bibliography entry using index 99>

<def ma &ref, –
   comment first place reference in text, –
   '(',par(1)(1),', ',par(1)(3),')', –
   comment now make index entry, –
   stack(xindex), xindex = 99, –
   x(0,,par(1)(1):par(1)(2),par(1)(3):'. ':par(2)),unstack(xindex),–
   edef ma &ref>

<comment &BIBHEAD and &BIBLIO format the index>

<def &bibhead = '<pageend>List of References<lend c, vs 4mm>'>
<xheaderlist(99) = '&bibhead'>

<def ma &biblio,–
   lend, i b 0, i h=1 l 1, vs 4mm, keep, –
   if par(1)=1, then, rep(12,'–'), ' ', –
   par(3), else, par(3), ' ', par(4), eif, –
   lend, endkeep, –
  edef ma &biblio>
<xmacrolist(99)='&biblio'>

<def &final = '<xindex=99,x(0)>', at eof &final>
<comment disable error about extra-long index entries>
<disable m 268>
```

```
There are several factors leading up to delinquency.
<&ref(&west,'<&refwest>')>
In spite of the fact that large cities have a high number of
delinquents, there is hope. <&ref(&smi,'<&refsmi>')>
Even Donald West does not dispute this point.
<&ref(&west,'<&refwest>')>
However, Smith backs up his findings. <&ref(&smi,'<&refsmi>')>
In his second study, West detailed the delinquent way of life.
<&ref(&west2,'<&refwest2>')>
Much was repeated from his previous study.
<&ref(&west,'<&refwest>')>
James West's earlier study was more original.
<&ref(&westj,'<&refwestj>')>
```

produces:

There are several factors leading up to delinquency. (West, 1973) In spite of the fact that large cities have a high number of delinquents, there is hope. (Smith, 1986) Even Donald West does not dispute this point. (West, 1973) However, Smith backs up his findings. (Smith, 1986) In his second study, West detailed the delinquent way of life. (West, 1977) Much was repeated from his previous study. (West, 1973) James West’s earlier study was more original. (West, 1927)

List of References

Smith, John Q. Urban Turmoil: The Politics of Hope. New City: Polis Publishing Co., 1986

West, Donald J. The Delinquent Way of Life. London: Heinemann, 1977

------------ Who Becomes Delinquent? London: Heinemann, 1973

West, James H. Who Becomes Delinquent In the Twenties? New York: Oxford, 1927

Controlling the Index Collection and Sorting

The INDEX command provides more control over the indices by letting you modify how the entries are collected and stored. The format of the command is:

```
<INDEX kwd . . . >
```

It affects only the index determined by the current value of XINDEX, so use the INDEX command after XINDEX is set. Any of the following kwds may appear in the command (more than one may be given):

ON
permits the collection of index entries. If INDEX ON is not specified, the first index entry encountered turns on indexing.

OFF
ignore all index entries made by X. If INDEX OFF appears before the first call to X, no index will be collected until an INDEX ON is encountered.

LEVELS = expression
will reset the maximum number of index levels to ‘expression’. The default value is 3, and the maximum is 8. If used, LEVELS must appear before the first index entry is saved. The smaller that LEVELS and MAXLEN are, the faster the index will run. If too many levels are used, the error is *Too many levels specified for index entry. Entry ignored.* These entries may not appear in exactly the right place in the Index, but their entire contents will appear.

MACROCOPY
The lines in the file –TXTFXSnn may be optionally stored in a file if the keyword MACROCOPY has been specified in the INDEX command for the index of the value of XINDEX. The file name would be –TXTFXMnn where ‘nn’ is the value of XINDEX. The default is –MACROCOPY.

MAXLEN = expression
will reset the maximum allowable length for each entry at each level to ‘expression’. The default is 64. If used, MAXLEN must appear before the first index entry is saved. The smaller that LEVELS and MAXLEN are, the faster the index will run.

```
<INDEX  MAXLEN  =  10>
<X(0,PNCTR,'extralongword')>
```

produces the error *Index entry too long to sort properly. Sorted entry may not appear in the proper place in the Index.*

ALL_LEVELS
If used, the output macro gets all the parameters even though the value of a parameter may be the same as the value of the same parameter of the previous entry.

```
<INDEX  ALL_LEVELS>
<X(0,1,’red’,’blue’)>
<X(0,2,’red’,’green’)>
```

produces:

```
red,  blue,  1
red,  green,  2
```

COLLATE = expression
controls the index sorting sequence. By default, the index entries are sorted according to the normal EBCDIC sorting sequence. If this is unsuitable, as it might be for text in language other than English, the sorting sequence can be changed with the following command:

```
<INDEX  COLLATE  =  –
#  ...  #:–
#  ...  #:–
#  ...  #>
```

where ‘...’ represents the new sorting sequence. It contains hexadecimal values of all the characters arrange in the order in which they are to be sorted. An example of a

sorting sequence suitable for the Russian language can be found in a file; see TXFM:INDEX for the location. The COLLATE sequence is applied to the data passed to the X and therefore should be given before the first use of X.

LINE DRAWING

The line drawing facility in TEXTFORM allows horizontal and vertical lines to be drawn.

Specifying Position of Line

The position of a line can be given by using the LOCATION command. This command allows a location be associated with any position on a page, within the dimensions of PAGESIZE. The syntax of the LOCATION command is:

<LOCATION [kwd] expression1 [kwd] expression2 location-id>

kwd
may be one of:

> **R**ELATIVE – causes the length following to be adjusted through vertical and/or horizontal justification.
> **A**BSOLUTE – causes the length following to be fixed at that point regardless of justification.

expression1
horizontal length measured from the top left corner of the page.

expression2
vertical length measured from the top left corner of the page.

location-id
is an alphanumeric id beginning with a letter that may be used in the DRAW command (see below) or as a parameter to LOCATION_INFO. Do not precede this name by ‘&’. After the page has been formatted, all location-ids become undefined, and may be used again. If the same location-id is defined more than once on a page, it generates the error *LOCATION ID is already defined. This one ignored.*

<LOCATION 1INCH 1INCH TOPLEFT>

places a location-id called TOPLEFT 1 inch from the top and left of the page.

<LOCATION PAGE_POSITION(1) PAGE_POSITION(2) HERE>

In this example, PAGE_POSITION is used to place a location-id at the current horizontal and vertical location.

Drawing lines

The DRAW command draws lines between two location-ids, using the current font, typesize, and underlining values, and whatever box drawing characters are available in the current character set. The resulting line must be either horizontal or vertical. Otherwise, you get the error message *Diagonal lines are not supported.* The syntax of the DRAW command is:

<DRAW [FROM] location-id1 [TO] location-id2 [structure-name]>

location-ids
are those defined by the LOCATION command before the end of the page.

structure-name

– if used, must be defined before the end of the page on which the actual drawing is to be done.
– if omitted or is given as '' then default drawing takes effect. Default drawing uses specific characters to draw the horizontal or vertical lines, depending on the current character set. These specific characters are generally the VBAR (vertical bar) for vertical lines and the DASH or LIGHTRL for horizontal lines. Default drawing automatically joins horizontal and vertical lines with joining characters (i.e. TLCORNER BLCORNER TRCORNER BRCORNER) if available. The lines are formatted in the typesize, character set, and font which is in effect when the DRAW command is encountered.
– if given, it is the name of a structure which contains drawing information to replace default drawing. The structure can contain five or six elements:

1) a string that will contain the characters of the line segment, as in &struc = ('*/', ... or &struc = (bullet, ...
2) a length that is the separation between string segments that make up the line. For example, if &struc = ('*/', .1in, ... then a horizontal line would look like
   */ */ */ */ */
3) a length giving the line thickness. Since most devices do not support line thickness this element in the structure should be a zero length, as in &struc = ('*', 0in, 0in, ...
4) the character set to use when the line is drawn, as in &struc = ('*', 0in, 0in, curcs, ... or &struc = ('*', 0in, 0in, 'optimist', ...
5) the font number in the character set above, as in &struc = ('*', 0in, 0in, curcs, 2)
6) the typesize (if supported) in which the line segment will appear. If the output device does not allow typesize changes, this structure element can be omitted.

For example:

<LOCATION 2IN PAGE_POSITION(2) LEFT>
<LOCATION 7.5IN PAGE_POSITION(2) RIGHT>
<DRAW FROM LEFT TO RIGHT>

produces:

```
<LOCATION 2IN PAGE_POSITION(2) LL>
<LOCATION 7.5IN PAGE_POSITION(2) RR>
<DEFINE &STARS = ('*', WORDSPACE, 0IN, CURCS, 1 )>
<DRAW FROM LL TO RR &STARS>
```

produces:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

The following examples show the difference between absolute and relative locations.

```
This <LOCATION A PAGE_POSITION(1) A
PAGE_POSITION(2)–LINESPACE LTBOX>
<LOCATION A PAGE_POSITION(1) A
PAGE_POSITION(2)+LINESPACE LBBOX>
sample
<LOCATION A PAGE_POSITION(1) A
PAGE_POSITION(2)–LINESPACE RTBOX>
<LOCATION A PAGE_POSITION(1) A
PAGE_POSITION(2)+LINESPACE RBBOX>
is a sentence. <LEND JUSTIFY>
<DRAW FROM LTBOX TO RTBOX>
<DRAW FROM LTBOX TO LBBOX>
<DRAW FROM LBBOX TO RBBOX>
<DRAW FROM RTBOX TO RBBOX>
```

produces:

This sample is a sentence.

```
This <LOCATION R PAGE_POSITION(1) R
PAGE_POSITION(2)–LINESPACE LTBOX2>
<LOCATION R PAGE_POSITION(1) R
PAGE_POSITION(2)+LINESPACE LBBOX2>
sample <LOCATION R PAGE_POSITION(1) R
PAGE_POSITION(2)–LINESPACE RTBOX2>
<LOCATION R PAGE_POSITION(1) R
PAGE_POSITION(2)+LINESPACE RBBOX2>
is a sentence. <LEND JUSTIFY>
<DRAW FROM LTBOX2 TO RTBOX2>
<DRAW FROM RTBOX2 TO RBBOX2>
<DRAW FROM RBBOX2 TO LBBOX2>
<DRAW FROM LBBOX2 TO LTBOX2>
```

produces:

This sample is a sentence.

Location and draw commands can be put into a macro if desired. However, **dynamic** location-id names should be used so that different locations are created each time the

macro is used. (A value that is calculated by a program, rather than inserted manually, is calculated dynamically.) The following example defines a macro that will draw a horizontal line from the left margin to the right margin
Note: Disable the warning message *'name' is not yet defined. Execution of 'name' deferred until macro is used* because you don't want the variable to be defined until the macro is called.

```
<DISABLE MESSAGE 228>
<DEFINE MACRO &HORZLINE>
  <&TOPLEFT = 'HXX':&CTR, &TOPRIGHT = 'HX':&CTR>
  <L>
  <LOCATION A LEFTMARGIN A PAGE_POSITION(2) $&TOPLEFT>
  <LOCATION A PAGESIZE(1)–RIGHTMARGIN A PAGE_POSITION(2)
$&TOPRIGHT>
  <DRAW FROM $&TOPLEFT $&TOPRIGHT>
  <&CTR=&CTR+1>
<EDEF MA &HORZLINE>

<DEF &TOPRIGHT,DEF &TOPLEFT>
<DEF &CTR=1>

<&HORZLINE>
```

produces:

Information of line position

LOCATION_INFO returns a structure containing information about the location-id given. It takes the name of the location-id as parameter. For example:

```
<LOCATION A 2IN A 4IN CURRENT>
<DEFINE &LINFO>
<&LINFO = LOCATION_INFO('CURRENT')>
<DEFUNITS = IN>
<DISPLAY &LINFO>
```

produces:

1 ABSOLUTE
2 2IN
3 ABSOLUTE
4 4IN
5 CURRENT

If the location-id given is undefined, the structure returned will contain one null element and FUNCTIONRC will be set to four.

The following example shows how to draw a box 1 inch from the top and bottom of the page, and halfway into the left and right margin as shown on this page:

```
<DEFINE MACRO &PAGEBOX>
<LOCATION A LEFTMARGIN/2 ABSOLUTE 1IN TOP_LEFT>
<LOCATION A PAGESIZE(1)–(RIGHTMARGIN/2) A 1IN TOP_RIGHT>
<DRAW FROM TOP_LEFT TO TOP_RIGHT>
<LOCATION A LEFTMARGIN/2 A PAGESIZE(2)–1IN BOT_LEFT>
<LOCATION A PAGESIZE(1)–(RIGHTMARGIN/2) A PAGESIZE(2)–1IN
BOT_RIGHT>
<DRAW FROM BOT_LEFT TO BOT_RIGHT>
<DRAW FROM TOP_LEFT TO BOT_LEFT>
<DRAW FROM TOP_RIGHT TO BOT_RIGHT>
<ENDDEFINE MACRO &PAGEBOX>

To draw the box:
<&PAGEBOX>
```

The AT ENDOFPAGE or AT STARTOFCOLUMN commands can be used if you want this box to appear repeatedly.

PROGRAM CONTROL WITH TEXTFORM

The programming aspects of TEXTFORM can be used to provide dynamic control over the format of a document. This section describes such facilities. Some have been used in examples in earlier parts of the manual.

System Variables

System variables are used by TEXTFORM to store information; they often change as the result of a command. You may display and use these variables for comparisons or tests, but you *cannot* change them. Only TEXTFORM may do this. System variables cannot be STACKed or UNSTACKed (to do this, you would be attempting to change them) but you can store their values in another variable. For example:

```
<DEFINE  &OLDT  =  TSIZE>
<TYPESIZE  12POINT>  . . .
<TYPESIZE  &OLDT>
```

System variables are often used in macros where they can be queried for information about the current run by using the IF command. A short list of the most important ones:

CURFONT – the current font
ODLOADED – becomes TRUE when the OUTPUTDEVICE command has been given
THISPAGE – alternates between LEFT and RIGHT, changing when page begins
LINDENT and RINDENT – length of current left or right indent
LINDENTINDEX and RINDENTINDEX – the index of current left or right index
PNCTR – the current page number

System Constants

These can be used like a system variable, (YEAR produces the 2010) but TEXTFORM sets them only once per run. Some of the most useful are:

TIME – time at the start of the TEXTFORM run (00:00.00 – 23:59.59)
DAY – current day of the week
DATE – current date of the month (from 1 to 31)
MONTH – current month
YEAR – current year

Information About Current Page Position

This section discusses ways to determine the position of text on the page through facilities in the TEXTFORM program, and how to use these results.

Four names are provided to determine the current position on the page of text. They are:

PAGE_POSITION
COLUMN_POSITION
POSITION
REMAINING

They differ in what they measure. PAGE_POSITION measures from the top left-hand corner of the physical page (sheet). COLUMN_POSITION measures from the top left-hand corner of the column, i.e., below the top page reserves if present. POSITION measures from top left-hand corner of the current float, reserve, footnote, or open text. (See TEXT_DESTINATION below). Normally this is measured from the bottom of the column reserves and the left indent setting. In abnormal cases, such as inside a FLOAT, the measurement is made from the top left-hand corner of the FLOAT.

REMAINING is the only name that measures forward. It measures towards the bottom right-hand corner of the float, reserve, footnote, or open text, telling how much space remains to insert text.

These items return horizontal and vertical information, depending on how they are used.

- If the parameter is 1, the horizontal position is returned.
- If the parameter is 2, the vertical position is returned.
- If no parameter is given, both horizontal and vertical positions are returned as a structure containing two lengths.

Since these are functions that return information, the information returned must be stored in a variable or used in a test. You cannot DISPLAY it. For example, to check if there is enough space for a 5 inches bottom float in the current column:

```
<IF  REMAINING(2)  >=  5IN,  THEN,  . . .
```

The following conditions may cause unexpected answers.

- Within a table, REMAINING is always very large (not the actual value)
- Inside of RESERVEs or FLOATs which have a size specified, the remaining vertical distance reflects the maximium size it can become, rather than the actual specified size.

TEXT_DESTINATION can be used to determine the type of item that is currently being built. It returns a one element structure containing a string. The string is one of TEXT (for open text), KEEP,† FOOTNOTE, FLOAT, or RESERVE.

----
†KEEP will be returned if a KEEP ENDKEEP sequence has been processed and the formatted line where TEXT_DESTINATION is executed is within the bounds of the KEEP.

Formatting Information

This section provides details about what TEXTFORM does at boundaries, and reviews formatting information in more detail. The part of the TEXTFORM program that controls page formatting decisions is called the **page processor**.

Pages

A page is started explicitly with the PAGE command, or by a command which forces TEXTFORM to start the page, such as NP or KEEP. (When text overflows a page, before TEXTFORM can place a character of open text on the next page, it starts the new page implicitly.) PNCTR and THISPAGE are changed immediately. Several ATs can occur at this point. The first is AT REFERENCE PAGEEND, (or AT REFERENCE PEND); the second is AT ENDOFPAGE.

When TEXTFORM starts a new page, it uses the size in PAGESIZE. The side of the page (LEFT or RIGHT) is determined by checking the values PRINTON and PNCTR. Four variables control the margins: they are TOPMARGIN, BOTMARGIN, LEFTMARGIN, and RIGHTMARGIN.

As all these variables are accessed and their values used, they are copied into a set of shadow variables. These variables are named CURPAGESIZE (which points to CURPAGEWIDTH and CURPAGEHEIGHT), CURTOPMARGIN, CURBOTMARGIN, CURLEFTMARGIN, and CURRIGHTMARGIN. The purpose of these variables is to allow two distinct questions to be asked: 1. What size is the page I am currently formatting? Check CURPAGESIZE. 2. What size will the next page be? Check PAGESIZE.

TEXTFORM then processes all the RESERVE PAGE, and FLOAT PAGE items that are pending for this page. While these items are being processed, the system variable CURCOL contains the value 0. When these items have been placed on the page, TEXTFORM knows the space remaining on the page for open text. Once the page has been started, space allocated for the bottom reserve cannot be recovered, even if the reserve is suspended before the bottom of the page is reached.

The page is now started. If the page overflow was caused by a KEEP, the part of the keep which appeared on the previous page is then copied to the current page. If the page overflow was caused by text, or if AT ENDOFPAGE or AT REFERENCE PAGEEND produced text, this text is now included in the open area of the page.

Any subsequent change to the size of the page, its margins, or its side, will not take effect until the next page. Changing those variables will have no effect on the page just started.

PAGE_POSITION can be used to determine the current position on the page of text. It measures from the top left-hand corner of the physical page (sheet) and returns two lengths: horizontal and vertical. In the following, &CHECK gives the PAGE_POSITION of this page:

```
<DEFINE  &CHECK  =  PAGE_POSITION>
<DISPLAY  &CHECK>
1     2.1458IN
2     9.972IN
```

If you want to check only the vertical position on the page:

```
<IF PAGE_POSITION(2) < 4IN, THEN, . . .
```

At the start of a run, PAGE_POSITION(2) is 0IN. After a PAGE command, which starts the page, its value reflects the space used for top margins, reserves, and floats. reserves, floats. When a page overflows, or when AT ENDOFPAGE becomes true, POSITION, COLUMN_POSITION and PAGE_POSITION indicate the next position where text can occur. REMAINING indicates the amount of horizontal and vertical space remaining at that point.

Columns

A column overflow, like the page overflow, causes something to start. However, the column overflow (or COLUMN or COLUMNEND commands) also have the power to start something other than a column. If the page processor is positioned between pages, the COLUMN command first must cause a page to be started before it can start a column. This is a direct result of having to know the limits of how tall and wide a column should be. It's impossible to know those limits until the size of the page, with all its margins and reserves, has been finalized.

The height of a column is the distance left after all the RESERVE PAGE and FLOAT PAGE things have been included. The width of the column, or perhaps even the number of the column, is determined by the LOGICALPAGE definition. If a logical page is not in use, the default, called PHYSICALPAGE, is used. PHYSICALPAGE has only one column, and has neither a LEFTGAP nor a RIGHTGAP. This makes PHYSICALPAGE the full size between the left and right margins in width, and between the top and bottom RESERVE PAGE or FLOAT PAGE items.

Once the size of the column is finalized, the value of CURCOL is adjusted to the column number, any RESERVE COLUMN and FLOAT COLUMN items are included, and the column is officially started. As a review,

1. if TEXTFORM is between pages, an implied PAGE command is done
2. the height of the column is determined by whatever is left over after the page is started
3. the width of the column is taken from the LOGICALPAGE definition
4. RESERVE COLUMN and FLOAT COLUMN items pending for this column are included

To get a blank page, with only a page number, and without any column reserves or floats on it, do

```
<PAGE,PAGEEND>
```

The page will be started, doing the page number, but no column will be started before the page is finished. Therefore, there won't be any column reserves or floats.

To get a page with page reserves and floats, and column reserves and floats, but no open text, do the following, which says "keep dumping columns until the end of the page":

```
<DEFINE &EMPTYCOL='<COLUMN,COLUMNEND>'>
<PAGE,&EMPTYCOL, comment that is the first one>
<WHILE 'CURCOL ¬= 1' DO &EMPTYCOL,Comment the rest>
```

COLUMN_POSITION can be used to determine the current position on the column. It measures from the top left-hand corner of column, i.e., below the top page reserves if present. As described on page 150, it returns two lengths: horizontal and vertical. On a one-column page, COLUMN_POSITION(1) gives the distance from the left margin.

```
<DEFINE &CHECK = COLUMN_POSITION, DISPLAY &CHECK>
1    0.8958IN
2    2.1388IN
```

When a column overflows, or AT ENDOFCOL becomes true, REMAINING, POSITION, PAGE_POSITION and COLUMN_POSITION reflect information about the next position where text can occur.

Lines

A line overflow, like the column and page overflow, requests that the next line be started. If the page processor is not currently processing a column, then an implied COLUMN command is generated. The LINE command has the same effect.

The horizontal limits of the line start out at the column boundaries, but are then reduced inwards by the indent settings. On the left edge, there is the left indent, LINDENT, and if this is the first line of a paragraph, then also the paragraph indent, PARAIND. On the right edge there is only the right indent, RINDENT to be subtracted.

The vertical position of the line is determined by LINESPACE. The value used is then stored in CURLINESPACE. If LINESPACE is changed in mid-line, it reflects the spacing that will be used for the *next* line while CURLINESPACE reflects the spacing for the *current* line. However, to ensure that text starts at the same position on each page, regardless of LINESPACE, the first line of each top reserve, top float, and the first line of each column of open text starts TSIZE instead of LINESPACE down from the bottom boundary of the preceding item. (For example, when using the page printer with TOPMARGIN=1IN and no reserves or floats, open text always starts on line 7, whether LINESPACE=.16IN or .33IN. This is because TSIZE is .1389IN and the device cannot change typesize. This is rounded to .16IN, which produces text on line 7.)

POSITION and REMAINING, described on page 150, can be used to determine the amount used and remaining on the current line.

Words

Words in the input are delimited by blank spaces, or by the beginning or end of the input line, or by the ALLOWLINEBREAK command. The current word being processed by TEXTFORM is contained in the system variable CURWORD. You can examine CURWORD but you cannot change it. CURWORD contains only the characters of the word up to the last character processed, as the example below

shows. After a blank is encountered, CURWORD is null.

If an input file contains the prime character available at the keyboard, rather than LQUOTE and RQUOTE, the following macro can examine CURWORD and produce the appropriate quote in the output. First, alter all primes in the file to &Q, so that the quoted words are <&Q>word<&Q>.

```
<DEFINE MACRO &Q, –
    IF CURWORD='', THEN, LQUOTE, ELSE, RQUOTE, ENDIF, –
ENDDEFINE MACRO &Q>
```

An AT ENDOFWORD command is available, as described on page 69. It lets actions be carried out at the end of every word.

When the characters in a word have variable widths, TEXTWIDTH can be used to determine the width of the word. POSITION provides the current position on the line.

Portions of words or strings can be obtained using the substringing facilities, described in the section on variables (page 98). For example, remove the word 'The ' from index entries with the following macro:

```
<define macro &x, –
    &w = par(1), –
    if uppercase(&w(1|4)) = 'THE ', –
    then, &w = &w( 5; chars(&w) ), –
    endif, –
    x(0,pnctr,&w), –
edef macro &x>

<define &w, define &len>
```

Blanks

In the input, a blank (or end of line) is a 'wordend' command. Multiple blanks are ignored (unless INPUTMODE is ASIS or PREFORMATTED):

- following another blank
- around any command which forces the end of a line and therefore the end of a word: LINEEND, COLUMNEND, PAGEEND, NEWPARA, VERTGAP
- Before exitting a state. These are essentially implied LINEEND commands: ENDFOOTNOTE, ENDFLOAT, ENDRESERVE, TAB, USE

Blanks do not affect the use of any commands other than INPUTMODE, BLANKCHARACTER, and INDENT RIGHT. Blanks are not required at the beginning or end of the input line. Errors are produced if blanks appear before or after the LOGICALBACKSPACE command or the ¬ character.

In the output, all words are separated by a blank space equal to WORDSPACE if ALIGNMENT=LEFT. Extra blank space must be produced via HORSPACE, BLANKCHARACTER, or SPLIT commands.

In command mode, blank spaces inside the initiator and terminator have no special meaning other than to delimit keywords, except in the assignment to the INPUTMODE variable. If text is generated from command mode, as in <'text string'> or <MONTH>, you must allow for blank spaces around the text. TEXTFORM does not insert a blank when it comes out of command mode, unless it also ends the input line at the same time, as follows:

<comment command>
text
<comment command>

In the following example, blanks are *not* automatically inserted before and after the text:

<comment command, –
'text', –
comment command>

A blank can be included in the string, as follows:

word<'text '>word

produces ‘wordtext word’. The sequence

<comment command, 'text text2'>

produces only one WORDSPACE between ‘text’ and ‘text2’ (unless INPUTMODE is ASIS or PREFORMATTED) regardless of how many blanks appear between the delimiters. To produce extra space, use HORSPACE or BLANKCHARACTER commands.

Sentences

The amount of space at sentence end is controlled by SENTSEP. Both of the following tests must succeed for TEXTFORM to recognize a sentence:

1. If a sentence end character . ! ? is the last character in the word, or is followed by) ' ’ or ”, and the character is not the only character in a word, then this is the end of a sentence.
2. If the first character of the word following a recognized sentence end is (' ‘ “ or a capital, then a SENTSEP space rather than a WORDSPACE is inserted.

If necessary, the command NOSENTENCE, or NS, can be used to suppress SENTSEP from being inserted. The command SENTENCE, or S, can be used when TEXTFORM does not recognize a sentence ending by the above methods.

Including Input If a Condition is True or False

The IF command causes text and/or commands to be included in the document, depending on the result of a test. The format of the command is:

```
<IF comparison [,] [THEN [,] body] [,ELSE [,] body2] , ENDIF>
```

Both 'THEN body' and 'ELSE body2' are optional, although one of them must appear. Note that the command separator must be included before the ENDIF, and before the ELSE if it is present. Any amount of text and/or commands may appear in 'body' or 'body2', including further IF commands.

The **comparison** is the logical test which determines which part of the IF ('body' or 'body2') to include. If the result of the comparison is TRUE, 'body' is included, otherwise 'body2' is included.

A missing ENDIF in a comparison produces, at the end of the run, the errors *Missing ENDIF for IF comparison in line xxx of source. End of file encountered while skipping text*.

When TEXTFORM is reading the input and looking for an ENDIF command, it does not evaluate the contents of variables or macros. For this reason, TEXTFORM would not find the ENDIF in the following example:

```
<DEFINE &TEST = '<IF VERSION=1.0, THEN>'>
<DEFINE &ETEST = '<ENDIF>'>
<&TEST> Include this text <&ETEST> and then continue
```

To make TEXTFORM immediately evaluate and act on the contents of &ETEST, use the execute operator:

```
<DEFINE &TEST = 'ENDIF' >
<&TEST> Include this text <$&ETEST> and then continue
```

How Comparisons are Performed

In a comparison, type only one less than sign, <. It is not necessary to enter it twice. The comparison has the form:

expression1 logical relation *expression2*

The two expressions on either side of the logical relation, often called *arguments*, are evaluated, then

1. converted to the same type
2. and compared. The comparison indicates that the first expression is <, =, or > than the second. According to this result, the logical relation which was specified in the comparison is evaluated as TRUE or FALSE.

The expressions on either side of the comparison are converted as follows:

expression 1	*expression 2*				
	hex	**number**	**scaled**	**length**	**string**
hex	hex	number	scaled	length	string
number	number	number	scaled	length	string
scaled	scaled	scaled	scaled	length	string
length	length	length	length	length	string
string	string	string	string	string	string

All logical relations are allowed:

=	(equal)
¬=	(not equal)
< or¬>=	(less; not greater or equal)
<= or ¬>	(less or equal; not greater)
> or ¬<=	(greater; not less or equal)
>= or ¬<	(greater or equal; not less)

The logical result of the comparison (< = >) is applied to the ‘logical relation’ specified. The logical comparisons which are allowed are:

comparison result *logical relation*	<	=	>
=	false	true	false
¬=	true	false	true
>= or ¬<	false	true	true
<= or ¬>	true	true	false
> or ¬<=	false	false	true
< or ¬>	true	false	false

The comparison is then flagged as either TRUE or FALSE according to the value in the above table.

Strings are compared ‘as is’:

'a' ¬= 'A' and
'a ' ¬= 'a'

Since the variable ODNAME always contains an upper case string, the following test will fail:

<if odname = 'x9700' . . .

If only one argument (of the comparison) is a string, then the other is converted to a string. If both arguments are numbers they are converted to the same type of number before comparison.

Structures are compared element by element. If the two structures being compared have a different number of elements, the structure with the most elements is considered greater than the smaller structure, regardless of its content.

When comparing lengths, you may want the test to succeed even though the lengths are not precisely the same. The FUZZ variable is provided for this purpose. If two arguments of a comparison are within FUZZ internal units, (one internal unit is 0.00001 millimeter), the two arguments will compare as equal. The default, which is:

```
<FUZZ = 125>
```

is equivalent to saying that all lengths within .00125mm of each other will be considered equal in comparisons. This is adequate in most cases, unless you want the following type of test to succeed:

```
<DEFINE &LENGTH = .3333IN*3>
<FUZZ = 300>
<IF &LENGTH = 1IN, THEN, 'TRUE', ELSE, 'FALSE', ENDIF>
```

produces:

```
TRUE
```

Including Input from a List of Choices

The CASE statement lets you *execute* any item (usually a command) in a list (structure) depending on the value of a given expression. It eliminates the need for IF commands within IF commands. The format of the command is:

```
<CASE expression kwd structure>
```

expression
must evaluate to a number greater than 0, and no greater than the number of elements in ‘structure’. If ‘expression’ does not fall into this range, it causes the error *CASE index out of range . . . Nothing executed.*

kwd
must be OF or FROM. Both have equivalent meaning.

structure
may be a structure, or the name of a structure, which contains strings. The element of ‘structure’ that is executed depends on the value of ‘expression’.

For example:

```
<CASE &TEST OF ( ' lend right ' , ' lend left ' )>
```

If &TEST is equal to 1, the command <lend right> will be done; if &TEST is 2, it will be <lend left>. In another example, you may have a macro that accepts one of four parameters, and you want to set a variable to TRUE, depending on the parameter supplied:

```
<case &zz+1 of ( 'error 8 ''illegal parameter to macro'' ', –
      '&flag1=true', –
      '&flag2=true', –
      '&flag3=true', –
      '&flag4=true') >
```

In this example, if &ZZ is 0 your error message appears. Otherwise, depending on

the value of &ZZ, the CASE statement sets one of the four variables to TRUE.

Including Input WHILE a Condition is True

The WHILE command includes the input specified depending on the success of a comparison:

<WHILE 'comparison' DO name>

comparison
is provided as a string. A component of ‘comparison’ must be changed in name to terminate the looping, otherwise the WHILE will go on forever (unless the comparison fails on the first try).

name
is done if the test described under ‘comparison’ succeeds. ‘name’ may be a variable, macro, command, or function. It may not contain parameters if it is a macro, function, or command; nor may it be substringed or subscripted if it is a variable.

<DEFINE &PRINT = '<&CNT, &CNT = &CNT+1, LEND>'>
<DEFINE &CNT=1>
<WHILE '&CNT<5' DO &PRINT>

produces:

1
2
3
4

Including Input FOR a Specific Number of Times

The FOR command includes a specified amount of input, depending on the value of a control variable. The format of the command is:

<FOR 'name=expression' UNTIL 'limit' [STEP 'step'] DO name2>

‘STEP 'step'’ may appear before ‘UNTIL 'limit'’ if desired.

name = expression
‘name’ may be subscripted or substringed. It must be defined before the FOR is issued. Note that the whole assignment (‘name = expression’) *must be provided as a string*. ‘name’ (the control variable) may be modified during execution of the FOR.

limit
is executed for each iteration of the loop (therefore it may contain an expression), testing its value against the value of ‘name’. If the value of ‘name’ is not greater than the value of the executed ‘limit’ (<=), the contents of ‘name’ are included as input.

step

(if specified), is added to the control variable after every execution of the specified input. ‘step’ may contain an expression. If ‘step’ is not specified, ‘1’ is used in its place.

name2
is included as input if the test described under ‘limit’ succeeds. Notice that name2 is not provided as a string.

To print all the parameters in a macro, regardless of number, the following commands may be used:

```
<define &a = 1>
<define &b = '<par(&a), lineend>'>
<def ma &printer>
   <for '&a=1' until nrpars do &b>
<edef ma &printer>
```

To print every second parameter in the macro, include STEP 2 in the command:

```
<for '&a=1' until nrpars step '2' do &b>
```

Functions

A **function** is a program which may be predefined or available in MTS. Functions supply information or perform operations that might otherwise be tedious or awkward to do. Information passed to a function is a parameter. Another way of stating this is that the function *takes* a parameter. A function may *return* a result, or a VALUE, which can often be used in a comparison or as input. This result is stored in the system variable R0. Most functions that have been described in this manual can be used in this manner, as in DEFINE &REM=REMAINING(2). Functions that do not return a result are TOC, X, and LISTING.

In MTS, after a program is run, a **return code** indicates the level of severity of any errors. In the same way, TEXTFORM stores the return code of each function called in FUNCTIONRC. return code from the last function called. A return code of 4 usually indicates invalid parameters were given to the function.

TEXTFORM Functions

Several functions are predefined in the language; some have already been used in examples. For a complete list, look in Index to TEXTFORM Language.

Using Functions Which are Not Part of TEXTFORM

It is necessary for TEXTFORM to communicate with subroutines and system subroutines. This is done by using functions. Most tasks that can be done in other programming languages can be done in TEXTFORM; however programming certain things in TEXTFORM can be either cumbersome, or expensive. This is also where functions are of value. To use a function in TEXTFORM, you **load** it; this lets you bring in an extra procedure which is used during the execution (running) of the

program. The LOAD command (described in detail in Appendix 1) defines

- what parameters a function takes and/or returns
- whether it can be used in an expression
- what its value is if it is used in an expression.

Two types of functions can be loaded. These are MTS system subroutines, and functions defined by the TEXTFORM supporters which are not part of the TEXTFORM language. At present, user-written functions cannot be loaded.

Loading System Subroutines as Functions

Several examples are given here.

The system subroutine GUSERID does not take any information. However, it does return a string in Register 1. LOAD the subroutine as follows:

```
<LOAD GUSERID &ID RETURNS R1 VALUE STRING>
```

The TEXTFORM function &ID now will store the signon ID in Register 1, i.e. the system variable R1, and also return a value to be used in assignments or comparisons:

```
<&ID>
```

produces:

```
TXTF
```

while the comparison:

```
<IF &ID = 'FORM' . . .
```

would fail.

If the word VALUE is omitted in the LOAD command, the signon ID is stored in R1, but no value is returned. Thus &ID could not be used in comparisons or assignments. Its only purpose would be to put a string in R1, which could then be examined.

```
<LOAD CMDNOE &CMD TAKES STYPE STRING NUMBER>
<&CMD('CONTROL *PRINT*',15)>
```

produces:

```
*PRINT* assigned receipt number 889271
```

The system subroutine CMDNOE issues an MTS command without echoing it. The function TAKES the command and the number of characters it contains.

Loading TEXTFORM-Supplied Functions

The functions described in the next sections are provided by the TEXTFORM group, but are not part of the TEXTFORM language. Therefore, they must be loaded before you can use them.

Making Copies of Variables and Macros

The function MACWRITE writes the contents of a variable or macro to a file or a logical I/O unit depending on the parameters passed. It should be loaded using the following command as an example:

```
<load macwrite &mw takes stype len string getval optional getval>
```

The parameters to MACWRITE are as follows:

1. The variable or macro name to write, within delimiters
2. The logical I/O unit, FDUB, or filename as a variable.
3. An optional string, either TEXT or DEF.

    a. for variables it must be TEXT.
    b. for macros

        1) DEF puts DEF MACRO name and EDEF MACRO name at the beginning and end of the macro.
        2) TEXT only writes the contents of the macro.

For macros a record is written for each line on which the macro was defined. A variable occupies one output line.

```
<load macwrite &mw takes stype len string getval optional getval>

<def macro &test>
This is a test macro.
<edef macro &test>
<def &file='–p'>
<&mw('&test',&file,'DEF')>
```

The above example writes the contents of macro &TEST into the file –P.

Superscript and Subscript Functions

The functions TOSUP and TOSUB convert all digits, plus and minus signs in the parameter to superscript or subscript characters. The rest of the characters are printed unchanged. These functions are useful only when the current character set contains SUP1, SUP2, etc. They are used as follows:

```
<load tosup &printsup takes stype len string
                      returns r0 ptr value len string>
<load tosub &printsub takes stype len string
                      returns r0 ptr value len string>

<&printsup('123')>
```

produces ^{123}.

Automatically Doubling Meta-characters

The functions ATOTEXT and TOTEXT deal with the meta-characters ¬, <, _, and @. ATOTEXT doubles these four meta-characters and the " character. For example:

```
<load atotext &atotext takes stype len string
                          returns r0 ptr value len string>
<define &p = 'Testing if @ works.' >
<x(0, 1, &atotext(&p) )>
```

makes the index entry with a doubled @, so that it appears correctly in the index.

TOTEXT doubles all meta-characters except the command initiator in the parameter passed.

```
<load totext &totext takes stype len string returns r0 ptr value len
string>
```

Obtaining Values of Logical I/O Units

The function FNAME can be used to determine the value of an I/O unit during the run. As a parameter, It takes a variable containing the name of the I/O unit, and writes what was assigned to the I/O unit in the variable. For example,

```
<load fname &filename takes stype getval returns stype getval>
<define &iounit = 'scards', &filename(&iounit), display &iounit >
```

displays the current SCARDS file name on SERCOM.

Calling TEXTFORM as a Subroutine

TEXTFORM may be called as a program, via RUN, or as a subroutine. More than one subroutine copy may be active at the same time.

Entry point names

TXOFRM for OLD:TEXTFORM
TXPFRM for *TEXTFORM
TXNFRM for NEW:TEXTFORM

Location

Either in the above files, or in some cases (*TEXTFORM and NEW:TEXTFORM) as a predefined low core symbol.

Calling Sequence

Program: STYPE

```
CALL  TXPFRM(PARLST)
```

where PARLST, is a two byte integer length followed by a string (the parameter list). TEXTFORM will use the SCARDS, SPRINT and SPUNCH assignments from the RUN command.

Subroutine: STYPE

```
CALL  TXPFRM(ID,FLAGS,SCAPAR,SPUPAR,SPRPAR)
```

where

ID is a four byte integer value used to identify the individual subroutine copies of TEXTFORM. (Note that recursive calls back to this copy of TXTFRM are not allowed and will cause a fatal return code.)

FLAGS
is four bytes of I/O flags for SCAPAR, SPUPAR, and SPRPAR (the fourth byte is unused). Each byte may have one of the following values

00 same as last call (ignore parameter)
01 revert to default (ignore parameter)
02 fdub (as returned by GETFD)
03 file name (a two byte length followed by the file name)
04 routine entry point (a subroutine to use instead of SCARDS, SPRINT or SPUNCH).
05 block (pointer to a block of storage, where the first word indicates the number of bytes remaining in the block, the second word is an address pointing to the next place to use in the block.) Lines within the block will be half-word aligned, and will be a two byte integer line length followed by the line itself.

Note that blocks used for SCAPAR will have their length decremented, and storage pointer increased (first two words of the block) as TEXTFORM reads the lines in the block.

Note that blocks used for SPUPAR or SPRPAR will have their length decremented and storage pointer increased as TEXTFORM writes lines into the block. If the block is too small to contain the next line to be written TEXTFORM will act as though SPRINT or SPUNCH had returned with a return code of 4 (and the line will not be written).
06 string (SCAPAR only; a two byte integer length followed by the data)

If FLAGS is a negative number, the copy of TEXTFORM denoted by ID will be stopped, producing the final page and listing and freeing all storage used by TEXTFORM. For example, to stop a copy of TEXTFORM use

```
CALL  TXPFRM(ID,-1)
```

SCAPAR
(the SCARDS parameter) On first call, if FLAGS(1) is STRING, the

parameter will be passed in the PAR= field.

SPUPAR
: (optional) the SPUNCH parameter

SPRPAR
: (optional) the SPRINT parameter

Return Codes

Program:

0 normal return or warnings only
4 minor errors
8 probable errors
12 definite errors
16 serious errors
20 fatal error
24 severe fatal error

Subroutine:

In addition to the return codes above
28 parameter error
32 recursive call attempted

Description

Program:

TEXTFORM will run to completion as though it were invoked by the RUN command.

Subroutine:

Since TEXTFORM itself contains an ENTRY card, if you wish to call TEXTFORM as a subroutine, your calling program must contain an ENTRY card and be loaded before TEXTFORM otherwise TEXTFORM will be given control.

Text is processed from SCARDS as described by FLAGS(1), and distributed as described by FLAGS(2) and FLAGS(3).

The return code is determined by TEXTFORM in the same way that it is when run as a program except that on each call the return code is set to zero on entry (i.e. the return code from each call reflects the errors for that call only). When the subroutine call is finished (by calling with FLAGS=-1), the return code is that for the whole run. If a fatal return code is given for any call, all future calls for this copy will be ignored, and the fatal return code will be re-issued.

Upon first call a copy of TEXTFORM denoted by ID will be started. This means that several copies may be active at any one time.

The operation of TEXTFORM is the same as in program mode, except that I/O is under program control.

If NOPROLOGFILE is desired, it must be specified in the initial call as part of the parameter list.

Attention interrupts and timer interrupts are inactive. Program interrupts will automatically generate a return code of 24.

LIST OFF NOTHING –COPY is the default when TEXTFORM is called as a subroutine.

TEXTFORM AND MTS

The RUN Command in Detail

TEXTFORM is invoked by the RUN command specifying *TEXTFORM as the object file. TEXTFORM uses a number of logical input/output units. The logical units used are:

SCARDS – is the logical unit containing the text and the commands to be formatted. SCARDS defaults to *SOURCE* if it is not specified. Input lines longer than 256 characters are truncated (this may cause errors if part of a command is truncated). Implicit catenation may be obtained by placing

$CONTINUE WITH fdname RETURN

beginning at column 1 within the input, where ‘fdname’ represents a file or device name.
SPRINT – is the logical unit where the statistics, listing, and proof output are written. See the description of LIST and PROOF to control this output.
SPUNCH – is the logical unit where the output document is written. SPUNCH has no default setting, so if you are at a terminal and do not specify SPUNCH in your RUN command, you will be prompted for it with *Unit SPUNCH was referenced but is not assigned. Enter a file/device name for it, ''CANCEL'', or ''HELP''* In a batch run, if you do not specify SPUNCH, *sink* is used if data will be meaningful, as with the 9700 and 1403 output devices. For batch runs where SPUNCH should usually be assigned to a file, but is not specified, the run is terminated. This applies to output devices such as the CALCOMP, 6670, APS5, etc.
SERCOM – defaults to *msink* if it is not specified. It receives the error listing, and messages generated by the GUSER Command Interface. The value of SERERRS controls error output.
GUSER – is the logical unit from which the GUSER Command Interface (GCI) reads its commands. GUSER defaults to *MSOURCE* if it is not specified.

Other input/output units referenced:

If you have a file named TXTF.PROLOG, TEXTFORM automatically reads commands (and text) from this file before reading from the logical I/O unit SCARDS.
TEXTFORM accepts commands from the PAR= field of the RUN command before processing commands and text from the file TXTF.PROLOG or the logical I/O unit SCARDS. The PAR= field must be the *last* item in the RUN command. Anything read from the PAR= field is *in command mode*. Once the PAR= field is processed TEXTFORM switches back to text mode. If you have a file named TXTF.PROLOG and *do not* want TEXTFORM to use this file, the *first* command in the PAR= field must be NOPROLOGFILE (or NOPROLOG or NOPR). If a LIST or CROSSREFERENCE command is supplied in the PAR= field, the settings will remain in effect for the entire run and any other LIST or CROSSREFERENCE commands will be ignored. The system variable RUNPAR,

which has no default value, can be used in the PAR field of the run command.

run *textform scards=*file* par=runpar=true
<IF RUNPAR = TRUE, . . .

Examples of the RUN command:

run *textform par=od 'x9700' 'univers'

Here input is supplied from the terminal because SCARDS defaults to *SOURCE*. You will be requested to specify SPUNCH with:

Unit SPUNCH was referenced but is not assigned.
Enter a file/device name for it, "CANCEL", or "HELP".

Respond with:

sink

if you want the output sent to the terminal (do this only when the output can be read at a terminal), or with a file name if you want the output stored in a file. SPRINT output will not be produced because SPRINT was not assigned.

run *textform scards=*file* spunch=*print*

Here text is read from the file FILE, and output is sent to a printer; use this form when the outputdevice is '1403'. When the outputdevice is 'X9700', use

run *textform scards=*file* spunch=-*t*
run *pagepr scards=-*t*

run *textform spunch=*OUT* par=noprolog

In this example input is supplied from the terminal, and output is stored in file OUT. Commands and text in the file TXTF.PROLOG are not processed.

run *textform scards=*file* par=list off

In this run, a listing will not be produced, regardless of LIST commands in the file. You will be prompted for SPUNCH if you are at a terminal. If submitted in a batch job, SPUNCH will be produced only if it will be meaningful on the printer.

Ending the TEXTFORM Run

At the end of the run, when TEXTFORM encounters an end-of-file, it prints the current output page, and then prints any unfinished logical pages, tables or floats. If SPRINT has been assigned, it produces a listing, statistics, and cross references. Two commands are also available to end the run.

In open text (as opposed to reserves, floats, etc.) you can use the STOP command to halt execution as though an end-of-file were encountered. The ABORT command causes the job to be terminated by calling the MTS subroutine ERROR; no listing or

statistics are produced. If you issue the MTS command RESTART after an ABORT command, TEXTFORM produces the listing and cross references that have been collected.

You may interrupt the run by using the command MTS, which returns you to MTS. Then, as long as you haven't typed a RUN, DEBUG, or UNLOAD command, you can give the MTS command RESTART, and TEXTFORM will continue where it left off in the run.

The system variable RC contains the return code which will be given to MTS at the end of the run. You can change its value directly, or change it with the ERROR command.

```
<ERROR 4 'Too many parameters to macro'>
<DISPLAY RC>
```

produces:

4

Errors Which End the TEXTFORM Run

Any error with a return code (RC) greater than 16 ends the TEXTFORM run. This is because TEXTFORM cannot determine the correct action to continue the run. Such errors are:

Table entry is longer than a page. Terminating.
RESERVES on page leave no room for text. Terminating.

System Information During the Run

During a TEXTFORM run, the function MTSCMD(par) will pass 'par' to MTS for interpretation, returning to TEXTFORM immediately after. The $ character is necessary in the command. The MTS commands RERUN, RUN, LOAD, and UNLOAD should not be used.

```
<MTSCMD('$restart sercom=*sink*')>
```

The function SYSCMD is the same as MTSCMD. The function SYSCMDNOECHO is also the same, but the command sent to MTS doesn't appear at your terminal.

Several functions provide information about resources being used. SHOWCPU returns the amount of CPU time in seconds used since the beginning of the run. SHOWVM returns the number of pages of virtual memory in use by this job.

During the run, the system variable SOURCELNR contains the line number of the last line read from SCARDS. This line number is in MTS internal form (i.e. times 1000). SOURCELNR contains –100000001 when AT ENDOFFILE becomes TRUE.

The system variable VERSION contains the version number of TEXTFORM that is being run. It can be checked before new or de-emphasized commands are used:

<IF VERSION >= '1.40' THEN, PRINTON=BOTH, ENDIF>

Output on SERCOM

Error Messages and Warnings

In this manual, various TEXTFORM error messages are mentioned. These messages give an error number, issue a message that explains the error, and have a return code. Warning messages are less severe, and often indicate an adjustment made by TEXTFORM.

These messages appear in the listing (on SPRINT), and, if SERERRS is TRUE, on SERCOM. SERERRS automatically becomes FALSE at the start of a batch run. If you do not want the messages to appear on SERCOM, set SERERRS to FALSE.

Individual messages can be suppressed and re-enabled with the DISABLE and ENABLE commands, although fatal errors (which have a return code greater than 16 and end the run) are always printed. The format of these commands is:

<DISABLE **M**ESSAGE expression . . . expression*n*>
<ENABLE **M**ESSAGE expression . . . expression*n*>

where ‘expression’ is replaced with the number of the message. For example, to disable the error

Error 186: Output Device is already loaded. Command ignored.

which might appear as the result of an OD command in the PAR field of the run command, include the following before the OD command in your file:

<DISABLE MESSAGE 186>

Although DISABLE causes error messages to be suppressed, and ignored in the ERROR count that appears in the listing, the return code in the system variable RC is still changed to reflect the seriousness of the errors.

You can generate your own error messages with the ERROR command. For example you might issue an error message if the proper number of parameters are not supplied to a macro:

<if nrpars >=2 then, –
error 4 'incorrect number of parameters', endif>

The format of the command is:

<**ERR**OR number [expression] >

number

If it is not larger already, the return code in the system variable RC is set to 'number'. 'number' should be a multiple of four. (If 'number' is greater than 16, TEXTFORM ends the run.)

expression
(if provided) will be issued as an error message if 'number' is not less than ERRORS (see the LIST command in Appendix 1). This message will appear in the source listing (on SPRINT), and on SERCOM (if SERERRS is TRUE). The message will be flagged with a % to indicate that it is user generated.

```
<define macro &head>
    <if nrpars > 2, then>
    <error 4 'Too many parameters to &HEAD'>
    <else, par(1), newline 2, par(2), endif>
<edef macro &head>
<&HEAD(1,2,3)>
```

produces

%Too many parameters to &HEAD

The ERROR command calls the function UERR. You can produce error messages by calling this function directly, instead of using the ERROR command.

```
<UERR( 4, 'Incorrect parameters' )>
```

'par1' is the return code to assign. It should be a multiple of four. 'par2' is the message to print.

The DISPLAY command

The DISPLAY command displays the requested information on SERCOM. It has been used in examples in this manual; it is also used by GCI. The format of the command is:

<**D**ISPLAY kwd>

'kwd' may be one of:

name
the name of any variable, macro, table or logical page.

ATTRIBUTES name
The attributes of the name will be displayed. This display is the same as that generated in the listing if LIST ATR has been specified.

QUEUE
The input queue will be displayed. The queue contains the current source line, and all lines which are suspended pending the execution of some other line in the queue. The place of suspension in each line is also indicated.

INPUT
the current input line, and the current position in it.

For information about the status or progress of your run, you can also display logical I/O units if you use the FNAME function described on page 163.

Output on SPRINT

The Proof

If your file contains a PROOFDEVICE command, the proof copy appears on SPRINT by default. When the proof device is the same as the output device, output will not appear on SPUNCH.

The Listing

A listing can be produced on SPRINT, at the end of a TEXTFORM run, if SPRINT is assigned in the RUN command, or if a LIST command is given in the PAR file of the RUN command. The listing includes the contents of the input file, and may also supply information about variables, macros, etc.

TEXTFORM saves the listing in a temporary file (–TXTFLIST), which it creates (size=10 pages) if necessary. If you create the file, TEXTFORM will not destroy it at the end of the run. The contents of the listing are controlled by the LIST command, which has the format:

<**L**IST kwd . . . >

For example, if you want only hyphenated words listed, include the command:

<LIST OFF HYPHENATION>

kwd
one or more ‘kwd’s may be specified, as listed in Appendix 1. Any ‘kwd’ prefixed by a – or ¬ will negate the effect of the ‘kwd’. Note that not all keywords may be prefixed by the – or ¬.

If a LIST command is supplied in the PAR= field of the RUN command, the settings remain in effect for the entire run and any other LIST commands are ignored.

Adding Information to the Listing

You can add your own information to the listing by using the LISTING function.

<LISTING('Start detailed proof-reading here.')>

Sample Listing

The following description refers to the sample listing which follows. The boldface number for each item appears on the sample.

1 The first line states which version of TEXTFORM was run (this is different for *TEXTFORM and NEW:TEXTFORM); the date; the signon ID which ran the TEXTFORM job; and that the job was terminal or batch, in which case the receipt number is displayed.

2 This line indicates whether the file TXTF.PROLOG was read, and how many lines it contained. The prolog file is never listed.

3 Internal Page and Internal Line are internal TEXTFORM values, stored in INTPAGENR and INTPAGELNR. Internal page is the page number of the current page. It may differ from PNCTR. Internal line is the line number on the page. It is increased by one with each line printed.

4 Source Line indicates the line in the source file where the input occurred (see SOURCELNR). A blank in this column means the input was generated, and not in the source file (for example, input from a macro).

5 These headings describe character flags which appear between the source line number and the input line. Not all of these flags appear in a default listing.

6 Under GCI in Control, four columns left of the input line:

! indicates that processing in this part of the input was being controlled by GCI.

7 Under Partial Use Flag, three columns left of the input line:

* indicates the line is resumed from a previously suspended line.

a blank indicates that the whole line is listed.

8 Under Input Use, two columns left of the input line, the character indicates what is done with the input:

% the input is being saved in a macro definition

– the input is being skipped as result of an IF command

a blank indicates the input is being processed normally.

9 Under Input Origin, the character indicates the type of input. It may be:

! GCI input (a line entered using the GCI INSERT command)

. PAR field input

\+ macro input

$ string input

a blank indicates input from the source file.

10 This section displays values which were assigned to the logical I/O units in the MTS RUN command.

11 Under Input Line, the source from the file is printed. See the LIST SOURCE command description.

```
(1) University of Alberta   TEXTFORM  Version: 1.0 (ov79)   Thursday November 15, 1979          ID: TXTF   JOB: TERMINAL
(2) OPTIONS Prolog file TXTF:TXTF.PROLOG used (0 lines).
                                    !!!!GCI in Control                              SERCOM=*MSINK*
                                    !***Partial Use Flag                             GUSER=*MSOURCE*
(3) Internal      Source  (4)       !*--Input Use                          (5)      SCARDS=-A          (10)
    Page  Line    Line              !*-+Input Origin                                SPRINT=-1
     #     #       #         (6)    !*-+                    Input Line (11)         SPUNCH=
     0     0       0                 .OD,PD>
     1     1       1         (15)    <list on values, crossreference on -commands -pvariables>
     1     1       2                 <pnctr=4>                                                  (20)
     1     1       3                 <define &a=1, define &b=@&a>
     1     1       4                 <define &c='font 2'>
     1     1       5                 <define macro &print,-
     1     1       6         (8)   % newline 2, par(1), newline,-
     1     1       7               % enddef macro &print>
     1     1       8                 <$&c>                                                      (17)
     1     1                        $font 2,-
     1     1       8         (7)   +  >
     1     1       9                 <if &a=2, then>
     1     1      10               - Quotations from Gulliver's Travels
     1     1      11               - <endif, font 1>
     1     1      12                 And he gave if for his opinion, that
     1     1      13                 whoever could make two ears of corn  or two blades
     1     2      14                 of grass to grow upon a spot of ground where only one
     1     3      15                 grew before, would deserve better of mankind,
     1     3      16                 and do more essential service to his country,               (18)
     1     4      17                 than the whole race of politicians put together.
     1     5      18                 <newpara> He had been eight years upon a project for extracting
     1     6      19                 sunbeams out of cucumbers, which were to be put in vials
     1     7      20                 hermetically sealed, and let out to warm the air in raw
     1     8      21                 inclement summers.
     1     9      22                 <&print('Jonathan Swift')>
   +1 &PRINT
     1     9                        +,-
(16) 1     9                 (9)    +newline 2, par(1), newline,-
     1    10                        $Jonathan Swift<-
   &PRINT
     1    11                        +-
   -1 &PRINT
     1    11      22              *  >

     Commands Executed           14
     Macros Executed              1
     Functions Executed           0
     Expressions Evaluated        4

     # of Controls               21
(12) # of Control Strings        10

     Number of Words             89
     Number of Sentences          2
     Text Lines Printed          10
     Text Pages Printed           1

     Number of Errors             0
     Number of Warnings           0
```

12 Total number of commands, variables, etc., used in the run are printed (see LIST TOTALS).

Commands Executed:	total number of commands and variable assignments
Macros Executed:	total number of macros called
Functions Executed:	total number of functions called
Expressions Executed:	total number of expressions evaluated (for example, in variable assignments)
# of Controls:	total number of commands, variables, strings, etc., used in command mode. This includes, for example, input generated as a result of AT-points or macros.
# of Control Strings:	total number of command initiators, including those generated by TEXTFORM
Number of Words	
Number of Sentences	
Text Lines Printed	
Text Pages Printed	
Number of Errors:	messages which set the return code (RC) to 4 or higher, except those that have been disabled. Note that messages which set the return code higher than 16 are fatal, and will cause the run to stop without processing any more input.
Number of Warnings	

13 Cross reference information, if collected, appears here. The collection of this information is controlled by the CROSSREFERENCE command, described in the next section.

14 Storage and run time statistics appear here. They indicate how much storage was used by the symbol table (it contains every item—predefined or user-defined—used in the TEXTFORM language, and its value and attributes (variable, command, macro, special character, function)), and the amount of storage used and CPU time spent in each phase of TEXTFORM. See the example in the cross reference sample.

In the sample listing given, extra list options have been requested to produce a more detailed listing.

15 LIST ON provides EXPANSION, EVALUATION, and SOURCE source listing options.

16 EXPANSION lists macro input, flagging it with +. In the Internal Page column, +2 &name indicates the macro is being entered, &name indicates a return to the macro (after evaluating a parameter or executing another macro), -2 &name indicates the end of the macro. A recursive macro or macros which call other macros might go down further levels, i.e. +2, +3, etc.

17 EVALUATION

18 SOURCE lists source lines; resumed lines are flagged with * (resumed means TEXTFORM finishes reading a source line after executing some other commands, such as the contents of a macro).

19 LIST VALUES lists the value of each variable or macro that appears in the cross reference listing. If the item has been erased, no value is listed. This is the same as doing a DISPLAY for every variable and macro which has not been erased.

20 CROSSREFERENCE ON: The cross references itemize the frequency and type of use of variables, macros, and commands. –COMMANDS and –PVARIABLES suppress the cross referencing of these items. See the CROSSREFERENCE command for a description of the options available.

Statistics About the TEXTFORM Run

TEXTFORM provides statistics on SPRINT at the end of a run that has been normally terminated, if SPRINT was given in the RUN command. (Some statistics can be suppressed. Refer to the previous discussion on the LIST command.) See items 12 and 14 in the preceding description of the sample listing.

Statistics about the run can be produced at any time by using STATS, which generates the statistics which appear after the source listing. If STATS is used with a parameter, the statistics are written on SERCOM, otherwise they are written in the source listing.

Cross References

Cross reference information indicates where items are defined, referenced, or erased, or assigned new values. Each item in the language can be given a cross reference attribute at definition time, or by using the ATTRIBUTE command. The two attributes, 'AXR' and 'NXR', mean 'always cross reference' and 'never cross reference'.

The format of the CROSSREFERENCE command is like the LIST command:

<**C**ROSS**R**EFERENCE kwd . . . >

'kwds' are listed in Appendix 1. Any 'kwd' prefixed by a – or ¬ will negate the effect of the 'kwd'. Note that not all may be prefixed by the – or ¬. If a CROSSREFERENCE command is supplied in the PAR= field of the RUN command, the settings will remain in effect for the entire run and any other CROSSREFERENCE commands will be ignored.

Cross reference information is not collected unless the command CROSSREFERENCE ON appears in a file, or unless an item has an 'AXR' attribute. The cross reference information is not printed unless CROSSREFERENCE ON or LIST ON (which includes CROSSREFERENCE) are given, or unless the item has an 'AXR' attribute. The cross reference listing is produced on SPRINT, after the source listing and statistics.

TEXTFORM writes the cross reference information into a temporary file (–TXTFXR1) just before producing it. TEXTFORM creates this file (size=10 pages) if necessary. If you create the file, TEXTFORM will not destroy it at the end of the run. Before the cross reference is generated, all the entries in the file –TXTFXR1 are sorted, and put into the file –TXTFXR2. This file is treated in the same manner as –TXTFXR1.

Each reference is indicated by source line number and type. After the references are printed, the item's current value is printed if it has not been erased. The reference types indicated are:

A	executed as a result of an AT-point
D	defined
E	erased
F	function parameter
M	macro parameter
R	referenced
T	used in a comparison
@	pointed to something else
$	executed
=	assigned to

Sample Cross References

20 CROSSREFERENCE ON: The cross references itemize the use, by type, of use of variables, macros, and commands. –COMMANDS and –PVARIABLES suppress the cross referencing of these items.

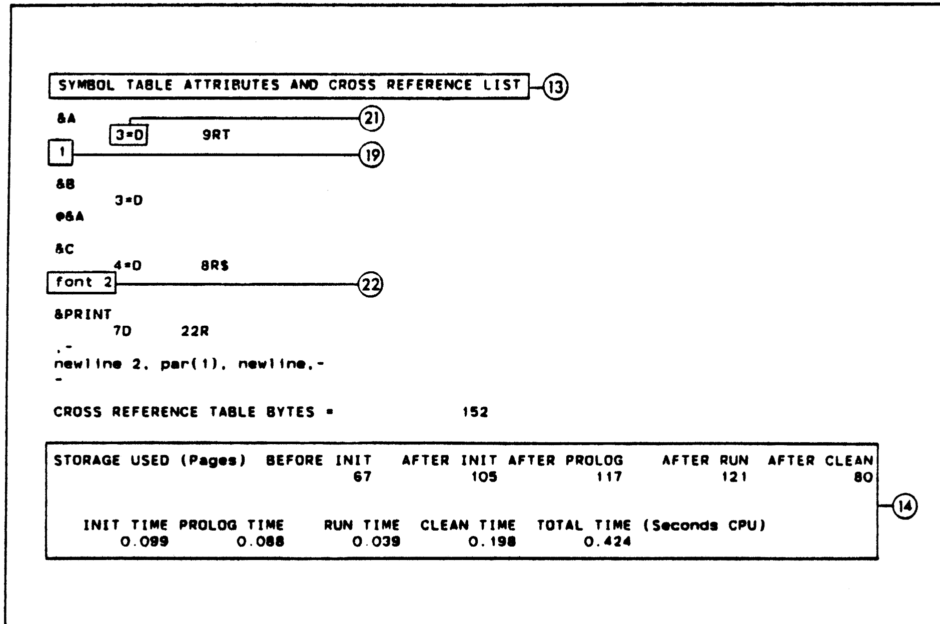
21 For each item, the type of reference is indicated with the source line number of its reference (the reference types are detailed under the CROSSREFERENCE command). For example, &A was defined (D) and had a value assigned to it (=) at line 3, was referenced (R) and used in a comparison (T) at line 9. &B was defined, but never referenced. &C was executed ($) at line 8.

22 LIST VALUES causes the value of each item to appear in the cross references, unless the item has been erased. For example, &C had the value 'font 2' at the end of the run.

Running TEXTFORM Interactively by Giving GUSER Input

The INTERACTIVE, or INT command causes a Guser Command Interface (GCI) interrupt. GCI is the same command as INT. TEXTFORM can then be run interactively from the terminal, as in the following example, where all the commands typed appear in italics. The GCI command can appear in a file, if you are running TEXTFORM at a terminal; it can be given after you hit ATTN; it or can be given in the PAR field of the RUN command.

\# *run *textform scards=test spunch=–1 par=gci*
15:02:49
! Guser Command Interface.
! Command ! *break 50*
! Command ! *run*
! At source line.
! Command! *display queue*
! 2 28 50 ! ! +,d q,font 2,par(1),f, –
|
! 2 28 50 ! ! +<&test>
|



! Command! *d lindent*
! 0MILLIMETRES
! Command! *d intpagelnr*
! 28
! Command ! *d &vertd*
! <nl,linespace = 8po,typesize 6po>
! Command ! *insert &vertd = '<lend,linespace =10po,typ 8po>'*
! Command ! *i list source*
! Command ! *st 20*
! Commands stepped
! Command ! *d &vertd*
! <lend,linespace=10po,typ 8po>
! Command ! *i list off*
! Command ! *i comment continue and then hit ATTN*
! Command ! *c*
WHAT? *gci*
! Attention Interrupt.
! Command!

When in interactive mode the following commands are accepted. Type the commands *immediately* after the space which follows the '!' character, as in

! Command! display input

Although you cannot insert blanks before the first command on the line, several commands may appear on one line:

```
! Command! display linespace, display alignment
```

RUN
same as CONTINUE

CONTINUE
to continue execution

STOP
to stop execution.

STEP number
If 'number' has a decimal point at the end of it, TEXTFORM will execute until that 'number' of source lines have been read, then it will cause a GCI interrupt; otherwise it will execute 'number' commands.

SKIP number
If 'number' has a decimal point at the end of it, TEXTFORM will skip 'number' lines of input; otherwise it will skip 'number' commands.

BREAK number
A GCI interrupt will occur if an input line having the line number 'number' is read. At this time 'number' cannot be negative.

INSERT body
'body' will be executed as though it had appeared in the input. 'body' is started after the blank after the INSERT command. 'body' is terminated by the end of the line. 'body' is assumed to start in command mode. A command separator is appended to the end of 'body'.

MTS
Return to MTS. TEXTFORM is RESTARTable, and will return to GCI for a new command.

ERROR [number]
Generate a GCI interrupt if an error is issued which has a return code of 'number' or greater. If 'number' is not specified, no GCI error interrupts will be issued.

DISPLAY pars
'pars' are any valid parameters for the DISPLAY command, described on page 171.

Interrupts in the TEXTFORM Run

A TEXTFORM run may be interrupted for several reasons. This section discusses how to continue after these interrupts.

Attention Interrupts

When you hit the ATTENTION key (BREAK on some terminals), TEXTFORM will respond:

WHAT?

The allowable responses are:

STOP or END-OF-FILE

The run is terminated as though an end-of-file were read, and the current output page is not produced. AT ENDOFFILEs are not done. Use AT REFERENCE STOP to have specific commands done after you use the STOP command.

CONTINUE, **CONT**

Continue from the point of interruption.

MTS or ATTENTION, or a CARRIAGE RETURN

Return to MTS, leaving TEXTFORM RESTARTable.

DUMP

Dump the contents of the registers at the point of interrupt. This is of interest only to the TEXTFORM support group.

GCI

Set a GCI interrupt. GCI will gain control at the next opportunity. For details, see GCI on page 177.

If you do not enter one of the above, TEXTFORM will respond with:

CONTINUE, STOP, MTS, DUMP OR GCI ARE VALID.
WHAT?

In some rare circumstances, TEXTFORM may not be interrupted. If this occurs, it will not sit there ignoring you, but print:

PLEASE WAIT

If you hit attention a second time, you will be returned to MTS. If you do not hit attention, TEXTFORM will get around to you as soon as it can.

The system variable ATTNS contains the number of attention interrupts which TEXTFORM has processed in this run.

Timer Interrupts

TEXTFORM contains a facility which interrupts execution every TIMERLIMIT seconds of CPU time, to ensure that endless loops or other errors do not use too many resources.† When the interrupt occurs, the current input line is printed on SERCOM, and you are asked whether you wish to continue.

```
TIMER INTERRUPT AT LINE . . .
display of current input line
DO YOU WISH TO CONTINUE?
```

The allowable responses are:

NO or end-of-file

The run is terminated as though an end-of-file were read. The current output page is lost.

YES

Continue.

MTS or a carriage return

Return to MTS, leaving TEXTFORM RESTARTable.

GCI

Set a GCI interrupt. GCI will gain control at the next opportunity.

If you do not enter one of the above, TEXTFORM will respond with:

```
ENTER YES, NO, MTS, OR GCI.
DO YOU WISH TO CONTINUE?
```

The timer interrupt message will not normally be encountered; however it may arise for one of the following reasons:

- during generation of a large index.
- during execution of a very large macro, or a very slow function. Respond YES a few times, unless you know that it shouldn't take this long; otherwise respond NO.
- during endless macro recursion, or a WHILE or FOR which will never terminate. Respond NO, if it is obvious that this is the case.
- due to a bug in TEXTFORM. Respond NO, if it is obvious that this is the case.
- due to an extremely long page which the output device processor is processing. This situation will most likely occur when using the CALCOMP output device.

If you have macros or functions which repeatedly cause timer interrupts, and wish to

----

†There are other time limits on a TEXTFORM run if 1) it is a batch run; 2) if there are global time limits for your signon id.

get rid of the nuisance (remember that you also get rid of the protection it offers), set TIMERLIMIT to a larger value, such as TIMERLIMIT=TIMERLIMIT*3. It is a good idea to only reset TIMERLIMIT where necessary; and not for the whole run. See an example of this in the index example on page 139.

Program Interrupts

Any program interrupts should be taken to the MTS consultants, preferrably before your source file is changed.

When a program interrupt occurs TEXTFORM will respond:

```
PGNT  xxxxxxxx  yyyyyyyy  type  module
NOW  WHAT?
```

Where 'xxxxxxxx yyyyyyyy' is the PSW (Program Status Word) at the time of the interrupt, 'type' is the interrupt type (e.g. PROTECTION etc.), and 'module' is the CSECT name of the interrupt (if it is a TEXTFORM CSECT). This information is of interest only to the TEXTFORM support staff. The allowable responses are:

STOP or end-of-file

The run is terminated as though an end-of-file were read. The current output page is lost.

CONTINUE, **CONT**

Continue from the point of interruption.

MTS or a carriage return

Return to MTS, leaving TEXTFORM RESTARTable.

DUMP

Dump the general registers at the point of interruption. If you take this action, the dump should be forwarded to the MTS consultants.

If you do not enter one of the above, TEXTFORM will respond with:

```
COMMAND  ERROR
CONTINUE,  STOP,  MTS,  DUMP  OR  GCI  ARE  VALID.
NOW  WHAT?
```

If you get a program interrupt, try entering CONTINUE several times. If the program interrupt still occurs, enter MTS and take your problem to the consultants.

If a program interrupt occurs during the processing of a program interrupt, MTS will handle it.

The variable PGNTLIMIT is the maximum number of program interrupts that TEXTFORM will allow in a batch job. Each time a program interrupt occurs, PGNTLIMIT is decremented. When it reaches zero, TEXTFORM terminates. The default number is 10.

PGNTS is a system variable containing the number of program interrupts that have occurred in the run.

APPENDIX 1 – TEXTFORM LANGUAGE

All of the predefined items that are part of the TEXTFORM language are listed in this appendix. If you encounter a predefined name that is not in this appendix, you can assume that it is a special character name.

– If you are unfamiliar with the notation used to describe commands, see page 32.
– Defaults are underlined. Abbreviations are **bold**.
– When [– | ¬] appears after a keyword, it indicates that the action of the keyword can be disabled by prefixing it with – or ¬.
– The type of item (command, variable, etc) appears on the right hand side of the column. When a page number is given here, it indicates where more information can be found about that item.

ABORT *Command 168*

ALB see ALLOWLINEBREAK *Command 22*

ALIGNMENT = kwd *Variable 11*

kwd

CENTER

CENTRE

JUSTIFY

RIGHT

LEFT

ALLOW**L**INE**B**REAK *Command 22*

ALPHABETIC(par) *Function 44*

ASIS *Constant 93*

AT kwd name *Command 69*

kwd

ASSIGN name2

END**O**F**C**OLUMN

END**O**F**F**ILE

END**O**F**L**INE

END**O**F**P**AGE

END**O**F**W**ORD

REFERENCE name2

START**O**F**C**OLUMN

TEXT**C**HARACTER c

c
a single character string, or
a special character name, or
a hex character entered as #FF#:" which is treated as a single character string.

AT_INFO(par1 [,par2]) *Function 72*

par1

ASSIGN name

END**O**F**C**OLUMN

END**O**F**F**ILE

END**O**F**L**INE

END**O**F**P**AGE

END**O**F**W**ORD

REFERENCE name

START**O**F**C**OLUMN

TEXT**C**HARACTER c

par2
required when ‘par1’ is ASSIGN, AS, REFERENCE, or REF, TEXTCHARACTER or TC.

ATT see ATTRIBUTE *Command 123*

ATTNS *System Variable 180*

ATTRIBUTE name kwd ... *Command 123*

name
is given the specified attribute(s)

kwd

The following attributes are valid for all items:

AXR

NXR

RESTRICTED

The following attributes are valid for variables:

CONSTANT

DISPLAY type
'type' may be any of the following (but only up to one from each group):

type (valid only for integer numbers or strings which can be converted to integer numbers):
ARABIC – digits
ENGLISH – words
FRENCH – words
ROMAN – roman numerals
ALPHABETIC – letters (a-z, aa-zz etc.)

which may be in (valid for all variables):
UPPER**C**ASE – upper case
LOWER**C**ASE – lower case

INCR = string3 (may not appear with LIST)

LIKE name2

LIST = structure

MA**X** = string4

MI**N** = string5

STRING

UPPER**C**ASE

LOWER**C**ASE

STATIC

DYNAMIC

FIXED

VARTYPE

The following attributes are valid for structures:

FIXEDELTYPE

FIXEDNRELS

A0 – A8 *Constant*

These are predefined values which can be used to produce metric pagesizes.

A0 = (33.1102IN,46.811IN)
A1 = (23.3858IN,33.1102IN)
A2 = (16.5354IN,23.3858IN)
A3 = (11.6929IN,16.5354IN)
A4 = (8.2677IN,11.6929IN)
A5 = (5.8268IN,8.2677IN)
A6 = (4.1339IN,5.8268IN)
A7 = (2.9134IN,4.1339IN)
A8 = (2.0472IN,2.9134IN)

BACK *Constant*

BC see BLANKCHARACTER *Command* *20*

BLANK**C**HARACTER [length] (= TYPESIZE) *Command* *20*

BLANKLINE = length (= 0.1667IN) *Variable*

This is the vertical size of a blank line. It is used when a NEWLINE command is used with a number greater than 1, which causes blank lines to appear on the page. This variable is being de-emphasized. Use VERTSPACE instead.

BOLD (= 3) *Constant* *38*

BOLDITALIC (= 4) *Constant* *38*

BOTH *Constant*

BOTMARGIN = length (= 1IN) *Variable* *46*

BOTTOM *Constant*

C see COLUMN *Command* *78*

CAP = logical value (= FALSE) *Variable*

When CAP is TRUE all output is capitalized.

<CAP = TRUE> forces all text following to be in upper case.
<CAP = FALSE> And some text following.

produces:

FORCES ALL TEXT FOLLOWING TO BE IN UPPER CASE.
And some text following.

CAPNEXTCHAR (= '@') *Constant* *30*

CASE expression kwd structure *Command* *158*

kwd

OF

FROM

structure
structure or name of structure

CEND see COLUMNEND *Command* *78*

CENTER *Constant*

CENTRE *Constant*

CHAP *Layout Macro*

CHARACTER**S**ET [expression] (= ODCHARACTERSET) *Command* *39*

CHAREXIST(par) *Function* *40*

par
a single character string
a special character name
a hex character entered as #FF#:'' which is treated as a single character string

CHARS(par) *Function* *98*

CHHRZ *Command*

This command is automatically executed by TEXTFORM whenever assignment is made to any horizontal length variable (e.g. WORDSPACE). You will never need to issue it.

CHPSZ *Command*

This command is automatically executed by TEXTFORM whenever assignment is made to PAGESIZE. You will never need to issue it.

CHVRT *Command*

This command is automatically executed by TEXTFORM whenever assignment is made to any vertical length variable (e.g. LINESPACE). You will never need to issue it.

COLUMN [expression] *Command* *78*

expression
the number of the column to start. If not given, the next column is started.

COLUMN_POSITION [(expression)] *Function* 153

expression

1
returns horizontal length measured from the top left corner of the column.

2
returns vertical length measured from the top left corner of the column.

COLUMN**END** [kwd] *Command* 78

kwd

TOP

BOTTOM

BOTH

CENTRE or **C**ENTER

JUSTIFY

COLUMNS = expression *Keyword* 76

COMDINT (= '<') *Constant* 26

COMDSEP (= ',') *Constant* 27

COMDTERM (= '>') *Constant* 26

COMMENT or **COMM** [body] *Command* 27

CONCL *Layout Macro*

CONTLINECHAR (= '-') *Constant* 27

CR see CROSSREFERENCE *Command* 176

CROSS**R**EFERENCE kwd ... *Command* 176

kwd

ON
will cause all items to be cross referenced. This keyword may not be prefixed by – or ¬.

OFF
will cause cross referencing to be shut off. This keyword may not be prefixed by – or ¬.

COMMANDS [– | ¬]
will cause the use of commands to be cross referenced.

MACROS [– | ¬]
will cause all macro calls to be cross referenced.

FUNCTIONS [– | ¬]
will cause all function calls to be cross referenced.

VARIABLES [– | ¬]
will cause the use of variables to be cross referenced.

PMACROS [– | ¬]
will cause all predefined macro calls to be cross referenced.

PFUNCTIONS [– | ¬]
will cause all predefined function calls to be cross referenced.

PVARIABLES [– | ¬]
will cause the use of predefined variables to be cross referenced.

&MACROS [– | ¬]
will cause all user-defined macro calls to be cross referenced.

&FUNCTIONS [– | ¬]
will cause all user-defined function calls to be cross referenced.

&VARIABLES [– | ¬]
will cause the use of user-defined variables to be cross referenced.

CS see CHARACTERSET *Command 39*

CTR1 = 1 (ATTRIBUTE CTR1 DISPLAY ARABIC I=1) *Variable*

CTR2 = 1 (ATTRIBUTE CTR2 DISPLAY ALPHABETIC LC I=1) *Variable*

CTR3 = 1 (ATTRIBUTE CTR3 DISPLAY ARABIC I=1) *Variable*

CTR4 = 1 (ATTRIBUTE CTR4 DISPLAY ALPHABETIC LC I=1) *Variable*

CTR5 = 1 (ATTRIBUTE CTR5 DISPLAY ROMAN LC I=1) *Variable*

CTR6 = 0 *Variable*

CTR7 = 0 *Variable*

CTR8 = 0 *Variable*

CTR9 = 0 *Variable*

CTR10 = 0 *Variable*

CURBOTMARGIN (= 1IN) *System Variable* *151*

CURCOL (= 1) *System Variable* *81*

CURCS (= DEFAULT) *System Variable* *39*

CUREMPFONT *Function* *38*

CURFONT (= 1) *System Variable* *38*

CURKEYBOARD (= STANDARD) see KEYBOARD *System Variable*

CURLEFTMARGIN (= 1IN) *System Variable* *151*

CURLINESPACE (= 0.1667IN) *System Variable* *153*

CURLP (= PHYSICALPAGE) *System Variable* *74*

CURPAGEHEIGHT (= 11IN) *System Variable* *151*

CURPAGESIZE (= (@CURPAGEWIDTH, @CURPAGEHEIGHT)) *System Variable* *151*

CURPAGEWIDTH (= 8.5IN) *System Variable* *151*

CURRIGHTMARGIN (= 1IN) *System Variable* *151*

CURTABLINE (= 0) *System Variable* *90*

CURTOPMARGIN (= 1IN) *System Variable* *151*

CURUNDERLINE (= 0) *System Variable* *37*

CURWORD *System Variable* *153*

D see DISPLAY *Command* *171*

DATE *Constant* *149*

DAY *Constant* *149*

DEF see DEFINE *Command* *95*

DEFCOLGAP = length (depends on table or page width) *Keyword* *77*

DEFCOLWIDTH = length (depends on table or page width) *Keyword* *77*

DEFINE [kwd] name [=expression] *Command* *95*

kwd

AXR

NXR

name
is the name of the variable to be defined. A user-defined name must be prefixed with an &, and may contain any of the characters a-z, _, &, or 0-9. The characters in the name may be in upper case, lower case, or both. The name may be of any length. 'name' may be subscripted (which will change the structure element of 'name' which has the specified subscript). 'name' may also refer to a substring, in which case a substring of 'name' will be changed. If the LIST ASTRACE command has been given, the new value of 'name' will be printed in the source listing.

expression
is the initial value to be assigned to the variable. It may be the result of any valid expression (including calls to functions which return a VALUE). If 'expression' is omitted, the variable is given a NULL string as its initial value. The contents of a variable are displayed by giving the variable name as a command.

DEFINE [kwd] **COL**UMN expression, keywords, E**ND****DEF**INE **COL**UMN expression *Command 91*

kwd

AXR

NXR

expression

number of column being defined

keywords

GAP = length (= DEFCOLGAP) *Keyword*

WIDTH = length (depends on table or page width) *Keyword*

ALIGNMENT = kwd (= ALIGNMENT) *Keyword*

FONT [expression] (= CURFONT) *Keyword*

COMMENT or **COMM** [body] *Keyword*

DEFINE [kwd] **L**OGICAL**P**AGE name, keywords, **E**ND**DEF**INE **L**OGICAL**P**AGE name *Command 76*

kwd

AXR

NXR

name

keywords

COLUMNS = expression (= 1) *Keyword*

DEFINE **COL**UMN expression, keywords,
END**DEF**INE COLUMN expression *Keyword*

DEFCOLWIDTH = length (depends on table or page width) *Keyword*

DEFCOLGAP = length (depends on table or page width) *Keyword*

LEFTGAP = length (= 0) *Keyword*

RIGHTGAP = length (= 0) *Keyword*

PAGESIZE = length structure (= CURPAGESIZE) *Keyword*

TOPMARGIN = length (= CURTOPMARGIN) *Keyword*

BOTMARGIN = length (= CURBOTMARGIN) *Keyword*

LEFTMARGIN = length (= CURLEFTMARGIN) *Keyword*

RIGHTMARGIN = length (= CURRIGHTMARGIN) *Keyword*

COMMENT or **COMM** [body] *Keyword*

DEFINE [kwd] **MA**CRO name ,[body,] **E**ND**DEF**INE **MA**CRO name
Command 113

kwd

AXR

NXR

name

body
any TEXTFORM input

DEFINE [kwd] **TA**BLE name, keywords, **E**ND**DEF**INE **TA**BLE name
Command 83

kwd

AXR

NXR

name

keywords

COLUMNS = expression (= 1) *Keyword*

DEFINE **COL**UMN expression, keywords, **E**ND**DEF**INE COLUMN expression *Keyword*

DEFCOLWIDTH = length (depends on column width) *Keyword*

DEFCOLGAP = length (depends on column width) *Keyword*

LEFTGAP = length (=0) *Keyword*

RIGHTGAP = length (= 0) *Keyword*

COMMENT or **COMM** [body] *Keyword*

DEFUNITS = kwd *Variable 31*

kwd

INCH, INCHES, IN
MILLIMETER, MILLIMETRE, MILLIMETERS, MILLIMETRES, MM
PICA, PICAS, PI
POINT, POINTS, PO
UNIT, UNITS, UN
LINES

DELAY [kwd] kwd2=expression name *Command 72*

kwd

VERTICAL

HORIZONTAL

kwd2

LENGTH

LINES

DISABLE **M**ESSAGE expression ... *Command 170*

DISPLAY kwd *Command 171*

kwd

name

ATTRIBUTES name

QUEUE

INPUT

DIV(par1, par2) *Function 111*

DRAW [FROM] location-id1 [TO] location-id2 [structure-name] *Command 145*

location-ids
are those defined by the LOCATION command before the end of the page.

structure-name

- if used, must be defined before the end of the page on which the actual drawing is to be done.
- if omitted or null, then default drawing takes effect. The lines are formatted in the typesize, character set and font which is in effect when the DRAW command is given.
- if given, it is the name of a structre which contains drawing indformation to replace default drawing. The structure can contain five or six elements:

  1) a string that will contain the characters of the line segment.
  2) a length that is the separation between string segments that make up the line.
  3) a length giving the line thickness. Since most devices do not support line thickness this element in the structure should be a zero length.
  4) the character set to use when the line is drawn.
  5) the font number in the character set above.
  6) the typesize (if supported) in which the line segment will appear. If the output device does not allow typesize changes, this structure element can be omitted.

EDEF see DEFINE *Command*

EEX *Layout Macro*

EFOOT see FOOTNOTE *Command 56*

EIF see IF *Command 156*

EFL see FLOAT *Command 60*

EKE see KEEP *Command 50*

ELSE see IF *Command 156*

EMPHNEXTCHAR (= '_') *Constant 38*

ENABLE **M**ESSAGE expression ... *Command 170*

ENDDEF see DEFINE *Command*

ENDDEFINE see DEFINE *Command*

END**FL**OAT see FLOAT *Command 60*

END**FOOT**NOTE see FOOTNOTE *Command 56*

ENDIF see IF *Command 156*

END**KE**EP see KEEP *Command 50*

ENGLISH(par) *Function 44*

EPT *Macro*

EPT stops lists of points. This resets all of the point levels until a new list of points is started. To stop points at any level and return to the previous level supply the number of the current level, as EPT(3), or simply use EPT(PTLEV):

```
<PT(1)>An item at level 1
<PT>Another item at level 1
<PT(2)>An item at level 2
<PT>Another item at level 2
<EPT(2)>
<PT>Last item at level 1
<EPT(1)>
```

produces:

1. An item at level 1
2. Another item at level 1
    a. An item at level 2
    b. Another item at level 2
3. Last item at level 1

An example of a paragraph within points:

```
<PT>An item at level 1
<PT>This is the beginning of a paragraph in level 1.
<PT(PTLEV+1)>This is the first point within the paragraph
<PT>This is the second point within the paragraph
<EPT(PTLEV)> This is the rest of the paragraph in level 1. It
will continue onto the next line. Notice the indentation and lack
of new counter. This is due to an EPT with no PT directly
following.
<PT>This is the last item at level 1
<EPT(1)>
```

This example produces:

1. An item at level 1
2. This is the beginning of a paragraph in level 1.
    a. This is the first point within the paragraph
    b. This is the second point within the paragraph
    This is the rest of the paragraph in level 1. It will continue onto the next line. Notice the indentation and lack of new counter. This is due to an EPT with no PT directly

following.
3. This is the last item at level 1

EQT *Layout Macro*

ER see ERASE *Command*

ERASE name *Command 122*

ERR see ERROR *Command 170*

ERROR number [expression] *Command 170*

EX *Layout Macro*

EXIST(name) *Function 122*

EXTRABOLD (= 5) *Constant 38*

EXTRABOLDITALIC (= 6) *Constant 38*

F see FONT *Command 38*

FALSE *Constant 102*

FL see FLOAT *Command 60*

FLOAT [kwd] [kwd2] [kwd3] [kwd4] [kwd5] name *Command 60*

kwd

VERY
when this keyword is used, the specified float will be the very first TOP or last BOTTOM float on the column or page. The first top float appears immediately after the last top reserve. Other floats may appear between VERY TOP and VERY BOTTOM floats (without the VERY keyword) if there is enough room for them on the column or page.

kwd2
indicates where the float will appear on the page. It may be one of:

TOP
at the top of the page or column. Input text which follows the float will appear on the current column, before the floated text.

BOTTOM
the float may appear at the bottom of the current page or column if there is enough space. If not, the float will appear at the bottom of the next page or column. ‘kwd5’, specifying the size of the float area, must be given for bottom floats.

kwd3
indicates if the floated text is to appear on a front (right) or back (left) page.

The float can appear only when PRINTON has the specified value. If this keyword is not specified, the FLOAT appears as soon as PAGE or COLUMN becomes true.

FRONT or **R**IGHT

BACK or **L**EFT

kwd4
if specified, it may be one of:

COLUMN
The system variable CURCOL can be used to ensure that text appears in a specific column:

```
   <DEF &FLO='<IF CURCOL=2,THEN,&COL,ELSE, FL T COL
&FLO, EIF>'>
   <FLOAT TOP COLUMN &FLO>
```

PAGE

kwd5

SIZE   horizontal length   vertical length
this specifies the size of the float, which will not appear until this size is available on a page. SIZE must fit within the open text area of the page, i.e. inside margins and reserves. If the size requested is larger than the text area on the page, the float will not appear (unless the text area is increased by changes to PAGESIZE, margins, or reserves). If SIZE is not given, a top float uses as much room as necessary to print the floated text (this may be more than one column or page). When SIZE is not given, there is no guarantee the float will all appear on the same page, since it may be the second or third float on the top float queue. A bottom float *must* have a size specified.

horizontal length
can be a horizontal length, or the word DEFAULT. If DEFAULT is given, it is the same as PAGESIZE(1)–LEFTMARGIN–RIGHTMARGIN. If the horizontal length given is wider than PAGESIZE(1), it produces the error *The RESERVE, or FLOAT is too wide. Maximum device width used.*

vertical length
is the vertical size the float will be, even if the float is empty. If the size specified is too small to contain all the text, TEXTFORM prints the error message *Vertical size specified in RESERVE or FLOAT is too small.* If it is a bottom float, this message also appears: *The size of this page may be wrong.*

These errors can be produced if you specify a vertical size of 12 points for the float, and TEXTFORM then tries to start the float in the default LINESPACE, which is 12.0507 points.

name

may be the name of an item (command, variable, macro, or table) containing the text and/or commands. ‘name’ is evaluated when the float is printed. If ‘name’ is a macro, parameters can be given if the entire macro call is treated as a string, as in:

```
<float top '&macrotop("Test")' >
```

However, the entire macro call is converted to upper case. To preserve lower case text:

```
<define &title = 'Test' >
<float top '&macrotop( &title )' >
```

If the FLOAT command is ended without specifying a ‘name’ to float, text may follow, ended by an ENDFLOAT command. In this case, text between the FLOAT and ENDFLOAT commands is evaluated immediately, then saved to be printed at the requested location. This form of the FLOAT command is not fully implemented and should not be used.

FLOAT_INFO [(par. . .)] *Function 67*

par

TOP

BOTTOM

FRONT

BACK

PAGE

COLUMN

name of float

FOGS kwd *Command*

This command controls the collection of readability information, and produces mathematical indices to measure the difficulty of your writing.

kwd

ON
If FOGS ON is included in the input, and HYPHENATION is ON with ALG in effect, data is collected.

OFF
data is not collected.

Note: The collection of FOGS information will increase the cost of the TEXTFORM run. This facility is not fully developed. The results provided in the current version may not be correct.

The readability indices are printed at the end of the run, after the source listing, and before the cross reference information, if the LIST HISTOGRAMS command appears in the input.

Readability is measured by:

1. Word distribution by number of syllables.
2. Sentence distribution by length.
3. Gunning's Fog Index provides the grade level of the document. Grade 9 is considered to be suitable reading. The index is calculated by the formula:
   SP + PS x 0.4 = School grade level, where:
   SP = Average sentence length
   PS = percentage of words of three or more syllables
4. Flesch's Readability Index uses average sentence length and the number of syllables per 100 words to calculate a reading difficulty index on a scale of 100. 24% is considered 'Difficult' (high school/college).

FONT [expression] (NORMAL) *Command* *38*

FOOTCONTINUE = expression (= "(cont'd)") *Variable* *57*

FOOTCTR = expression (= 1) *Variable* *57*

FOOTDIVIDE = expression (= '') *Variable* *57*

FOOTINDEX = expression (= 1) *Variable* *58*

FOOT see FOOTNOTE *Command* *56*

FOOTNOTE [body] **E**ND**FOOT**NOTE *Command* *56*

FOOTSEP = expression (= "<REP(18,'-'), LEND>") *Variable* *57*

FOR 'name = expression' UNTIL string [STEP string2] DO name2 *Command* *159*

FORDIS(name) *Function* *131*

FRENCH(par) *Function* *44*

FRONT *Constant*

FUNCTIONRC *System Variable* *160*

FUZZ = expression (= 125) *Variable* *157*

GALLEY = logical value (= TRUE) *Variable*

During the TEXTFORM run, the value of GALLEY determines whether each page is printed. Although it may be changed any time in a run, TEXTFORM only checks it at the end of each page, before the page is sent to SPUNCH.

GAP = length (= DEFCOLGAP) *Keyword* *91*

GARBAGECOUNT *System Variable* *236*

GARBAGESPACE *System Variable* *236*

GCI see INTERACTIVE *Command* *177*

HEAD *Layout Macro*

HEXDELIM (= '#') *Constant* *102*

HOR**S**PACE length *Command* *19*

HS see HORSPACE *Command* *19*

HYPHENATION kwd ... *Command* *23*

kwd

ON

OFF

DICT [=expression] [– | ¬]
–DICT or ¬DICT will shut off dictionary lookup (in this case ‘expression’ may not be specified). If used, DICT must appear with the ON keyword. If ‘expression’ is not provided and a dictionary has not been read then TEXTFORM’s dictionary in the file *TXTFHYPHDICT is used.

expression
if provided, specifies the name of the file containing the hyphenation dictionary to use. ‘expression’ may be a string of explicitly catenated filenames.

A hyphenation dictionary can be supplied in two forms—external and internal. The external form is described on page 24. If two or more dictionary file names are specified, the internal form dictionary must precede the external forms in the list of file names. Only one internal form dictionary file can be used in the list of names.

An internal form dictionary is created as a result of the WRITEDICT keyword. This file must *not* be changed in any way. Large dictionaries should be kept in internal form to save costs. Since only one internal form dictionary file can be used, you may want to combine TEXTFORM’s hyphenation dictionary with your own. If your hyphenation dictionary is in the file MYWORDS, create a file called INTDICT and then insert the following HYPHENATION command in your TEXTFORM file:

<HYPHENATION ON
DICT='*TXTFHYPHDICT+MYWORDS' –

WRITEDICT='INTDICT'>

Subsequent runs would then use the command:

<HYPHEN ON DICT='INTDICT'>

ALGORITHM [=expression] [– | ¬]

expression

ENGLISH

FRENCH

WRITEDICT = expression
causes the current dictionary that has been read to be written to the file specified by 'expression', as illustrated in earlier examples. *The file is emptied before the current dictionary is written.*

ERASEDICT
causes the storage occupied by the resident dictionary associated with the current ALGORITHM to be released.

TRY [– | ¬]
this is the same keyword as ALGORITHM.

READDICT = expression
the same as DICT=expression.

I see INDENT *Command 13*

IF comparison [,] [THEN [,] body] [, ELSE [,] body2] , ENDIF *Command 156*

IN *Constant 100*

INC see INCLUDE *Command 63*

INCH *Constant 100*

INCHES *Constant 100*

INCLUDE kwd [kwd2] [kwd3] [kwd4] [name] *Command 63*

kwd

FLOAT

kwd2

TOP

BOTTOM

kwd3

FRONT or **R**IGHT

BACK or **L**EFT

kwd4

COLUMN

PAGE

INDENT [**H**ANG = expression] kwd *Command 13*

[HANG = expression]
The change does not take effect until 'expression' formatted lines have appeared.

kwd

LEFT number2
The INDENT LEFT command has an immediate action of ending the current word and positioning at the specified left indent for the next word. LINDENTINDEX is changed to contain 'number2', and LINDENT is changed to contain the length in LEFTINDENTS(number2). The specified left indent then becomes the **current left margin**, which depends on the left margin of the page, the left indent, and the position of the current column on the page. Each line of text after a LINE command or line overflow begins at the current left margin. The INDENT LEFT command forces a new line to be started if:

– the line has been started, the specified indent is to the left of the previous indent (if INDENT LEFT 1 is in effect, LINE, INDENT LEFT 0 produces an empty line)
– the specified indent leaves no room for text on the current line (perhaps a right indent is also in effect)
– the current position is less than one device unit to the specified indent

If the specified indent does not cause a new line to begin, the text which appears on the line between the old and new left indents is aligned. The current value of ALIGNMENT is used, although alignments of BOTH or JUSTIFY are overridden to LEFT (similar to the LINEEND command).

RIGHT number3
When an INDENT RIGHT command is used, it does not end the current word, but changes the **current right margin**. The current right margin depends on the right margin, right indents, and the position of the current column on the page. It is the last (farthest right) position on the line where TEXTFORM will place text. Any word which will not fit on the line, before the current right margin, will begin on a new line at the current left margin. If the word contains ALB's or HYPHENATION is ON, the word will be broken at the closest point before the end of line; the first part being placed on the current line, and the last part being placed on the next line. As a result of the INDENT RIGHT command, RINDENTINDEX is changed to contain

'number3', and RINDENT is changed to contain the length in LEFTINDENTS(number3).

<LINE,INDENT BOTH 0>1. <INDENT BOTH 1 2>This is a heading which is several lines long. This is a heading which is several lines long.
<INDENT RIGHT 0,SPLIT '.'> 33

produces:

1. This is a heading which is several lines long. This is a heading which is several lines long. 33

BOTH number4 [number5]

JUSTIFY number4 [number5]

OFF

INDEX kwd . . . *Command 141*

kwd

ALL_LEVELS

COLLATE = expression

ON

OFF

LEVELS = expression (= 3)

MACROCOPY [– | ¬]

MAXLEN = expression (= 64)

INPUTMODE = kwd *Variable 93*

kwd

<u>UNFORMATTED</u>

PREFORMATTED

ASIS

INT see INTERACTIVE *Command 177*

INTERACTIVE *Command 177*

RUN or **RU**

CONTINUE or **CONT**

STOP

STEP number

SKIP number

BREAK number

INSERT body

MTS or CARRIAGE RETURN

ERROR [number]

DISPLAY pars

INTPAGELNR	*System Variable*	*173*
INTPAGENR	*System Variable*	*173*
INTRO	*Layout Macro*	
ITALIC (= 2)	*Constant*	*38*
ITYPE(name, [par2])	*Function*	*127*
JUSTIFY	*Constant*	
K see KEYBOARD	*Command*	
KEEP	*Command*	*50*
KEYBOARD [name]	*Command*	

name

STAN**D**ARD

APL1

APL2

If the character set being used is APL (via the OD or CS command), APL characters can be produced in a document by typing the special character name. However, the KEYBOARD command provides an alternate way of entering the special characters into a document.

Keyboard APL1 is used for DECwriter-type terminals. APL2 is used for Lektromedia, IBM 3270 or AJ510 terminals. After TEXTFORM encounters a KEYBOARD command in the file, it converts each output character to the alternate character that appeared on the key of the specified keyboard. The input text is not altered. If the current character set is APL and a KEYBOARD command is issued, all lower case alphabetic letters in the input will appear as APL letters and uppercase letters are translated to the

corresponding APL character on that key at the terminal. For example:

> The function to accomplish this is
> <LINEEND, KEYBOARD APL2> a <LEFTA> Q5 <K>

produces:

> The function to accomplish this is
> $A \leftarrow ?5$

To end keyboard translation, use KEYBOARD or
KEYBOARD STANDARD.

The procedure for using APL character sets differs slightly between the 1403 line printer and the X9700 page printer.

For OUTPUTDEVICE '1403' 'APL':
- the text appears with normal letters
- special character names can be used to produce individual APL characters, or the KEYBOARD command can be used as described above.

For OUTPUTDEVICE 'X9700':
- use character set TN when you want normal letters
- to produce individual APL characters, type:

  <CS 'APL', UPA, CS 'TN'>

  or the commands CS 'APL' *and* KEYBOARD name can be used as described above.

When using an alternate keyboard, note the following:
- the translation takes place only in text mode. Commands are not affected. Thus, the special TEXTFORM meta-characters <, @, ¬, and _ are treated as commands and are not translated.
- The series of characters <<, @@, etc. do not indicate special operators. They are therefore treated as text, and are translated.
- Characters that are produced from command mode by naming them are never translated. For example the command:

  <DAGGER>

  will produce that character regardless of the keyboard currently in effect.
- The system variable CURKEYBOARD is a string containing the name of the current keyboard.

L see LAYOUT *Command*

LAYNAME *System Variable*

LAYOUT name *Command*

LBS see LOGICALBACKSPACE *Command 41*

LC (= '<CAP=FALSE>') *Variable*

LEFT *Constant*

LEFTGAP = length (=0) *Keyword* *77*

LEFTINDENTS = length structure (= (.4IN, .8IN, . . . , 8IN)) *Variable* *13*

LEFTMARGIN = length (= 1IN) *Variable* *46*

LEND see LINEEND *Command* *12*

LI *Constant* *100*

LIGHT (= 7) *Constant* *38*

LIGHTITALIC (= 8) *Constant* *38*

LINDENT *System Variable* *18*

LINDENTINDEX *System Variable* *18*

LINE *Command* *5*

LINE**END** [kwd] *Command* *12*

kwd

- **C**ENTRE or **C**ENTER
- **L**EFT
- **R**IGHT
- **B**OTH
- **J**USTIFY

LINEREM *Function*

This function is de-emphasized. REMAINING(1) provides the same information.

LINES *Constant* *100*

LINESPACE = length (= 0.1667IN) *Variable* *6*

LINEUSED *Function*

This function is de-emphasized. COLUMN_POSITION(1) provides the same information.

LIST kwd ... *Command 172*

kwd

COMMANDS
List lines having commands.

EXPANSION
List macro input (listing flagged with +). In addition the macro nesting level is shown when a macro is entered:

+macro name nesting level

and when it is left:

–macro name nesting level

If a macro exits back to a previous macro, the name of the macro returned to is also printed:

macro name

EVALUATION
List executed strings (listing flagged with $).

SOURCE
List source lines (resumed lines are flagged with * in listing).

ERRORS [kwd2]

kwd2

TERSE
only the message number, source line and column number are printed. Neither the text of the message, nor the input text are printed. If the error is encountered in generated text or a macro, that fact is indicated.

NORMAL

VERBOSE
NORMAL, plus the message severity and a traceback through the input, like DISPLAY QUEUE.

ON
List all input (note that TRACE information is not included). This keyword may not be prefixed by – or ¬.

OFF
Do not list any input (note that this does not affect the TRACE listing). This keyword may not be prefixed by – or ¬.

ASTRACE
(ASsignment TRACE) List the new value (of the assigned variable)

after an assignment.

COM**PT**RACE
(COMParison TRACE) List the result of the two expressions used in every comparison, and the result of the comparison.

PAR**TR**ACE
(PARameter TRACE) list the value of each macro parameter used during the execution of a macro.

TRACE
ASTRACE, COMPTRACE and PARTRACE above.

ATT**R**IBUTES
List the attributes of each item, whether it has been cross referenced or not. This listing appears as part of the cross reference listing.

VALUES
List the value of each variable or macro that appears in the cross reference listing. If the item has been ERASEd, no value will be listed.

NOTHING
The listing does not appear, but is still collected, according to the other keywords given to the LIST command, in the file –TXTFLIST.

SOMETHING
The listing is restored.

CROSSREFERENCE or **XREF**
List the cross reference information which has been collected.

TOTALS
total of the number of commands, macros, etc. executed during the run will be printed after the source listing. Word, sentence, and error counts will also be printed.

SYMTAB
list symbol table statistics after the source listing. These indicate how much storage was used by the symbol table (it contains every item—predefined or user-defined— used in the TEXTFORM language, and its value and attributes (variable, command, macro, special character, function), how efficiently the storage was used, and how efficient the 'look-up' mechanism was.

HISTOGRAMS
list collected fogs data, including histograms for both word and sentence length. The statistics only reflect the data collected while FOGS was in effect.

STATISTIC**S**
same as HIST, SYMTAB and TOTALS combined.

HYPHENATION
list all words hyphenated and the hyphenation point, algorithm or

dictionary file used, and line number on page, page number, SCARDS line number. LIST OFF HYPHENATION suppresses all listings except the hyphenation listing.

COPY
File –TXTFLIST is copied to SPRINT and then emptied.

LISTING(par) *Function 172*

LOAD [kwd] string name [F=string2] [TAKES] [RETURNS] *Command 160*

This command loads a subroutine with the name ‘string’, giving it the name ‘name’ in TEXTFORM. It is loaded from the file ‘string2’ (if specified), or the TEXTFORM library if not.

kwd
The following additional attributes may be specified:

AXR
The function will have the ‘always cross reference’ attribute.

NXR
The function will have the ‘never cross reference’ attribute.

string
the CSECT name to load

name
the TEXTFORM name to define

F=string2
the file containing the function. Before any function is loaded, a check is made to see if it is currently loaded. If so, the loaded copy is used; otherwise, the function is loaded from the library specified.

TAKES kwd2
The parameter types for TAKES are the same as those for RETURNS below. If there is a TAKES parameter list, it must be described before the RETURNS parameter list.

RETURNS kwd2
‘kwd2’ may be any of the following parameter types:

R0 [PTR] [VALUE] kwd2
Register 0 is used. This may not be specified if TTYPE or NAME are used as well.

PTR
If PTR appears, the register will contain the address of the parameter.

VALUE
VALUE may only appear in the RETURNS list. If VALUE does appear, the parameter will be returned as the result of the

function. This type of function may be used in expressions. You can use the value returned from the function in an assignment, or, if you place the function call in the input, its value will appear in the output. If a function which does not return a VALUE is used in an expression a NULL string will be returned on its behalf.

kwd2
is the type of the parameter, as follows:

[kwd3] NUMBER
If 'kwd3' appears it must be one of BYTE, SHORT (half-word), or LONG (double-word). If 'kwd3' does not appear, WORD is assumed. If LONG appears, PTR must have appeared.

[LEN] STRING [L = expression]
If LEN appears, PTR must have appeared. If LEN appears, the string pointed to is a half-word string length, followed by the string. If 'expression' appears, and is greater than 4, PTR must have appeared. If neither LEN nor 'expression' appear, and PTR appears, a length of 256 is assumed.

LENGTH
A length is passed

SCALED
A scaled number is passed. To convert from scaled to real, divide by 10000.

GETVAL
PTR must have appeared. The parameter is passed in TEXTFORM's internal form.

THING
whatever is passed. If PTR is not coded the first 4 bytes of the parameter are passed in the register. For RETURNS, the PTYPE used will be that of the parameter.

R1 [PTR] [VALUE] kwd2
Register 1 is used. This may not be specified if STYPE or TTYPE are used as well. The types of R1 parameters allowed are the same as those for R0 above.

STYPE [VALUE] kwd2
An IBM S-type linkage is used. This may not be specified if R1, NAME or TTYPE are used as well.

VALUE
VALUE may only appear in the RETURNS list. If VALUE does appear, the parameter will be returned as the result of the function. This type of function may be used in expressions. If a function which does not return a VALUE is used in an expression a NULL string will be returned on its behalf.

'kwd2' is the type of the parameter, as follows:

[kwd3] NUMBER
If ‘kwd3’ appears it must be one of BYTE, SHORT (half-word), or LONG (double-word). If ‘kwd3’ does not appear WORD is assumed.

[LEN] STRING [L = expression]
If LEN appears, the string pointed to is a half-word string length, followed by the string. If neither LEN nor ‘expression’ appear, a length of 256 is assumed.

LENGTH
A length is passed.

SCALED
A scaled number is passed. To convert from scaled to real, divide by 10000.

GETVAL
The parameter is passed in TEXTFORM’s internal form.

THING
whatever is passed. For RETURNS, the PTYPE used will be that of the parameter.

TTYPE
A TEXTFORM t-type linkage is used. This may not be specified if R0, R1, NAME or STYPE are used as well. The TTYPE linkage is:

- register 0 – the length of the parameter list pointed to by register 1.
- register 1 – points to a half-word aligned list of parameters of the form; half-word parameter length followed by the parameter. Each parameter is half-word aligned. The whole list is terminated by a zero length parameter. True zero length parameters are flagged by a length of –1.

The allowable parameter types are the same as for an STYPE list.

NAME
A TTYPE parameter list containing the name of the parameter will be passed. This may not be specified if R0, R1, STYPE, or TTYPE are used as well.

```
<LOAD CMDNOE &CMD TAKES STYPE STRING
NUMBER>
<&CMD('CONTROL *PRINT*',15)>
```

produces:

```
*PRINT* assigned receipt number 889271
```

The system subroutine CMDNOE issues an MTS command without echoing it. The function TAKES the command and the number of characters it contains.

```
<LOAD GUSERID &ID RETURNS R0 VALUE STRING>
```

<&ID>

produces:

BOZO

The system subroutine GUSERID does not TAKE any information. In this example, a TEXTFORM function is loaded with the name &ID. It RETURNS a string in Register 1, which is BOZO if that was the current signon id when the job was run.

LOCAL [kwd] name [=expression] *Command 120*

LOCATION [kwd] expression1 [kwd] expression2 location-id *Command 144*

kwd

RELATIVE

ABSOLUTE

expression1
horizontal length measured from the top left corner of the page

expression2
vertical length measured from the top left corner of the page

location-id
name (containing letters or numbers) beginning with a letter. An & is not necessary. After the page has been formatted, all location-ids become undefined and may be used again.

LOCATION_INFO(par) *Function 144*

LOGICAL**B**ACK**S**PACE *Command 41*

LOWERCASE(par) *Function 44*

MACFLAG (= 0) *System Variable 118*

MAX(par1,par2) *Function 106*

par1

'**S**TRING'

'**L**ENGTH'

'**N**UMBER'
For instance, if par1 is 'LENGTH', then, every element in the structure specified by par2 is treated as lengths. Conversion is done automatically if the element was not given as a length. When comparing lengths, the result depends on DEFUNITS that is in effect at that time.

par2
a structure.

MAXWORDSPACE = length (= 5IN) *Variable 12*

MEMBER(par,structure) *Function 105*

MILLIMETER *Constant 100*

MILLIMETERS *Constant 100*

MILLIMETRE *Constant 100*

MILLIMETRES *Constant 100*

MIN(par1,par2) *Function 106*

par1

'**S**TRING'

'**L**ENGTH'

'**N**UMBER'
For instance, if par1 is 'LENGTH', then, every element in the structure specified by par2 is treated as lengths. Conversion is done automatically if the element was not given as a length. When comparing lengths, the result depends on DEFUNITS that is in effect at that time.

par2
a structure.

MINHYPH = expression (= 2) *Variable 24*

MM *Constant 100*

MONTH *Constant 149*

MTS *Command 169*

MTSCMD(par) *Function 169*

NC see NEWCOL *Command*

NEW**C**OL [expression] *Command*

Begins a new column. This command is de-emphasized. See COLUMN.

NEW**L**INE [expression] [kwd] *Command*

Begins a new line. This command is de-emphasized. See LINE.

NEW**PAGE** *Command*

Begins a new page. This command is de-emphasized. See PAGE.

NEW**P**ARA [length] [length2] *Command* *8*

length (= PARAIND)

length2 (= PARASEP)

NL see NEWLINE *Command*

NO (= 0) *Constant* *102*

NONLBWDSPACE (= '¬') *Constant* *29*

NOPROLOGFILE or **NOPROLOG** *Command* *167*

NORMAL (= 1) *Constant* *38*

NO**S**ENTENCE *Command* *155*

NP see NEWPARA *Command* *8*

NPAGE see NEWPAGE *Command*

NRPARS *Function* *117*

NS see NOSENTENCE *Command* *155*

OD see OUTPUTDEVICE *Command* *34*

ODCHARACTERSET = expression (= 'DEFAULT') see OUTPUTDEVICE *Variable*

ODFONT = expression (= 1) see OUTPUTDEVICE *Variable*

ODLOADED (= TRUE) see OUTPUTDEVICE *System Variable*

ODNAME = expression (= 'DEFAULT') see OUTPUTDEVICE *Variable*

ODTSIZE = length (= 0.1389IN) see OUTPUTDEVICE *Variable*

OFF (= 0) *Constant* *102*

OLDPT *Macro*

If OLDPT is specified, TEXTFORM will use the PT macro defined before version 1.4. In this version of PT, PTSEP controls the vertical spacing between points. e.g. when PTSEP=2, each point is separated by a NL 2 command.

ON (= 1) *Constant 102*

OUTPUT**D**EVICE [expression [expression2 [expression3 [expression4]]]]
Command 34

This command determines the output device. It must be given before text is encountered, or before a command containing a length is given. Even a command such as PAGE, which must check the PAGESIZE lengths, will cause the output device to be loaded.

expression
indicates the output device. If not given, ODNAME is used. If 'expression' is given, ODNAME is changed to contain 'expression'. It is stored as an upper case string. After the OUTPUTDEVICE command, ODNAME becomes CONSTANT.

expression2
the name of the output device character set to use at the start of the run. If it is not specified, ODCHARACTERSET is used. If 'expression2' is given, ODCHARACTERSET is set to 'expression2'. It is stored as an upper case string. ODCHARACTERSET can be changed during the run. It is the character set used by the default state. (If 'expression2' is given, then 'expression' must appear.)

expression3
the typesize to use at the start of the run. If it is not specified, ODTSIZE is used. If 'expression3' is given, ODTSIZE is set to 'expression3'. ODTSIZE can be changed during the run. It is the typesize used by the default state. (If 'expression3' is given, then 'expression' and 'expression2' must appear.)

expression4
the font number to use at the start of the run. If it is not specified, ODFONT is used. If 'expression4' is given, ODFONT is set to 'expression4'. ODFONT can be changed during the run. It is the font used by the default state. (If 'expression4' is given, the first three expressions must appear.)

Once the output device is loaded, the system variable ODLOADED becomes TRUE. Any attempt to re-issue the OUTPUTDEVICE command after this produces the error *Output Device is already loaded. Command ignored.*

During the TEXTFORM run, the value of GALLEY at the end of each page determines whether the page is sent to SPUNCH.

OVERSTRIKE expression *Command*

overstrikes characters in 'expression' in the output. The overstruck characters are centred on the largest of the group. This command is de-emphasized. Use LOGICALBACKSPACE instead.

<OVERSTRIKE '0-'>

produces:

θ

P see PAGE *Command 47*

PAGE *Command 47*

PAGE**END** *Command 47*

PAGENUM (= (@PNHEAD, @PNCTR, @PNTRAIL)) *Constant 49*

PAGENUM is composed of pointers to the three predefined variables: PNHEAD (the page number header), PNCTR (the automatically incremented page counter), and PNTRAIL (the page number trailer). PAGENUM is a constant, but because its elements are pointers to other variables, you may assign values to the elements (but not the whole structure).

PAGE_POSITION [(expression)] *Function 151*

expression

1
returns horizontal position measured from the top left corner of the physical page.

2
returns vertical position measured from the top left corner of the physical page.

PAGEREM *Function*

This function is de-emphasized. REMAINING(2) provides the same information.

PAGESIZE = length structure (= (8.5IN,11IN)) *Variable 46*

PAGEUSED *Function*

This function is de-emphasized. Use PAGE_POSITION(1) instead.

PAR (expression) *Command 113*

PARA**E**ND *Command*

This command ends the current paragraph. It has the same effect as LINEEND, and is provided only for consistency.

PARAIND = length (= 0.5IN) *Variable 8*

PARASEP = length (= 0.1667IN) *Variable 8*

PART *Layout Macro*

PD see PROOFDEVICE *Command 35*

PDCHARACTERSET = expression (= 'DEFAULT') see PROOFDEVICE *Variable* *35*

PDLOADED (= TRUE) see PROOFDEVICE *System Variable* *35*

PDNAME = expression (= 'DEFAULT') see PROOFDEVICE *Variable* *35*

PE see PARAEND *Command*

PEND see PAGEEND *Command* *47*

PGNTLIMIT = expression (= 10) *Variable* *183*

PGNTS *System Variable* *183*

PI *Constant* *100*

PICA *Constant* *100*

PICAS *Constant* *100*

PNCTR = expression (= 0) *Variable* *49*

PNHEAD = expression (= '') see PAGENUM *Variable*

PNTRAIL = expression (= '') see PAGENUM *Variable*

PO *Constant* *100*

POINT *Constant* *100*

POINTS *Constant* *100*

POSITION [(expression)] *Function* *150*

expression

1
returns horizontal position measured from the top left corner of the current float, reserve, footnote, or open text.

2
returns vertical position measured from the top left corner of the current float, reserve, footnote, or open text.

PREFORMATTED *Constant* *93*

PRINTON = kwd *Variable* *48*

kwd

BOTH

RIGHT

LEFT

FRONT

BACK

PROOF = logical value (= TRUE) *Variable*

Once the proof device is loaded, the value of PROOF determines whether a proof is produced. If PROOF = TRUE proof output is produced. The value of PROOF may be changed as many times as desired in a run, however the production of each page of proofed text depends on the value of PROOF at the end of the page, just before it is sent to SPRINT.

PROOF**D**EVICE [expression [expression2]] *Command 35*

This command works in the same way as the OUTPUTDEVICE command. The associated variables are:
PDNAME
PDCHARACTERSET
PDLOADED
PROOF
PDNAME is the proof device, stored as an upper case string;
PDCHARACTERSET is the proof character set, stored as an upper case string.
At the beginning of a TEXTFORM run, the proof device is not loaded. To load it, use the PROOFDEVICE command *after the OUTPUTDEVICE has been loaded*:
Once the proof device is loaded, PDLOADED is set to TRUE, and the value of PROOF determines whether a proof is produced.

PT [(level [,expression, . . .])] *Macro*

The PT macro provides an automatic facility for producing enumerated lists with indents. Parameters to the macro, described below, modify the action of the PT macro on a short-term basis only.

level
is the level of indent to use for the PT. Use PTLEV to change this dynamically.

expression
may be provided to change the counter for the current level. This remains in effect only until EPT appears for that level. More than one ‘expression’ can be provided. ‘expression’ may be any of the following:

'string'
replace the counter with this string until EPT is given for this level.

```
<PT(1,'*')>This is an item at level 1
<PT>Another item at level 1
<PT(2,'#')>This is an item at level 2
<PT>Another item at level 2
<PT(1,'*')>This is the third item at level 1
<PT(2,'#')>Last item at level 2
<PT(1,'*')>Last item at level 1
```

```
<EPT(1)>
```

produces:

```
* This is an item at level 1
* Another item at level 1
   #    This is an item at level 2
   #    Another item at level 2
* This is the third item at level 1
   #    Last item at level 2
* Last item at level 1
```

HANG
no counter is used until EPT appears for this level. The items are set off by hanging indents.

NUMBER
forces the counter to be used. This is most useful when the default points are not already numbered points, as when using PT('string') or PT(HANG). PT(NUMBER) also allows you to add characters to the front of the counter by saying PT('n',NUMBER).

```
<PT('A',NUMBER)>An item at level 1
<PT>This is the beginning of a paragraph in level 1.
<PT(PTLEV+1,NUMBER)>This is the first point within
the paragraph
<PT>This is the second point within the paragraph
<EPT(PTLEV)>
<PT>This is the last item at level 1
<EPT(1)>
```

produces:

```
A1 An item at level 1
A2 This is the beginning of a paragraph in level 1.
   a.   This is the first point within the paragraph
   b.   This is the second point within the paragraph
A3 This is the last item at level 1
```

Global changes to the action of the PT macro are made by changing related variables and structures:

PTITEMGAP PTPREGAP PTPOSTGAP	control vertical space around points
PTCOUNTER – CTRn PTPRESTRING PTPOSTSTRING	control the appearance of the counters
PTPRELEVELSTRING PTPOSTLEVELSTRING	can be used to automatically include input before or after specific levels of points.

PTPOSTINDENT PTTEXTINDENT PTPREINDENT PTTEXTINDENTHANG	determine indents used by points
PTSAVE PTRESTORE	save and restore changes to the above variables.

When you use the PT macro, as in:

```
. . . this is some text preceding the list.
<pt(1)> This is first enumerated point.
<pt(1)> This is second enumerated point.
<ept> and back to normal text . . .
```

the commands done are:

```
. . . this is some text preceding the list.
< ptpostindent(1) = lindentindex >
< vertgap ptpregap(1) >
< ptprelevelstring(1) >
< indent left ptpreindent(1) >
< ptprestring(1)>
< ptcounter(1) >
< ptpoststring(1)>
< indent h=pttextindenthang(1) left pttextindent(1) >
This is first enumerated point.
< vertgap ptitemgap(1) >
< indent left ptpreindent(1) >
< ptprestring(1)>
< ptcounter(1) >
< ptpoststring(1)>
< indent h=pttextindenthang(1) left pttextindent(1) >
This is second enumerated point.
< vertgap ptpostgap(1) >
< ptpostlevelstring(1) >
< indent left ptpostindent(1) >
```

PTCOUNTER = structure (= (@CTR1,@CTR2,. . .,@CTR10)) *Variable*

this structure contains the counters used by PT. The counters at the various levels can also be accessed by the variable CTR*n*, where *n* is the level.

```
<PTPRESTRING(1) = '*'>
<ATTRIBUTE CTR1 DISPLAY ALPHABETIC UPPERCASE>
<PTPOSTSTRING(1) = '*'>
<ATTRIBUTE CTR2 DISPLAY ARABIC>
<CTR3 = 0>
<PTPOSTSTRING(3) = '(#)'>
<PT>This is an item at level 1
<PT>Another item at level 1
<PT(2)>This is an item at level 2
<PT>Another item at level 2
<PT(3)>This is an item at level 3
```

```
<EPT(1)>
```

produces:

```
*A* This is an item at level 1
*B* Another item at level 1
     1. This is an item at level 2
     2. Another item at level 2
        (#)    This is an item at level 3
```

PTITEMGAP = length structure (= (0IN,0IN,. . .,0IN)) *Variable*

is the vertical space between items at the same level of points. Along with PTPREGAP, the vertical space before a level of points, and PTPOSTGAP, the space after a level of points, PTITEMGAP controls the space between points.

These GAP structures can be changed for all levels, as in:

```
<PTITEMGAP=PTITEMGAP+4MM>
```

or for individual levels of points, as in

```
<PTPREGAP(1)=4MM,PTPOSTGAP(1)=4MM>
```

which produces 4MM of space before and after level 1 points only. PTPREGAP=4MM causes an error because PTPREGAP must always be a structure with 10 lengths.

```
<LINESPACE=4MM, PTPREGAP = PTPREGAP+4MM>
<PTPOSTGAP = PTPOSTGAP+4MM>
<PTITEMGAP = PTITEMGAP+4MM>
<PT(HANG)>Adams, Robert. "Langland and the Liturgy
Revisited." Studies in Phililogy 73(1976):266-284.
<PT(HANG)>Alford, John Alexander. "A Note on Piers Plowman
B.xviii. 390: 'Til Parce it Hote'." Modern Philology
69(1972):323-325.
<PT(HANG)>Alford, John Alexander. "Some Unidentified
Quotations in Piers Plowman." Modern Philology 72(1975):390-399.
<EPT>
```

produces:

Adams, Robert. “Langland and the Liturgy Revisited.” Studies in Phililogy 73(1976):266-284.

Alford, John Alexander. “A Note on Piers Plowman B.xviii. 390: ‘Til Parce it Hote’.” Modern Philology 69(1972):323-325.

Alford, John Alexander. “Some Unidentified Quotations in Piers Plowman.” Modern Philology 72(1975):390-399.

PTLEV = expression (= 0) *Variable*

The variable PTLEV contains the PT level. It allows you to change levels **dynamically** without knowing the current PT level. A value that is calculated

by a program, rather than inserted manually, is calculated dynamically. For example:

```
<PT(PTLEV+1)>An item at level 1
<PT>Another item at level 1
<PT(PTLEV+1)>An item at level 2
<PT>Another item at level 2
<EPT(1)>
```

produces:

```
1.  An item at level 1
2.  Another item at level 1
    a. An item at level 2
    b. Another item at level 2
```

PTPOSTGAP = length structure (= (0IN,0IN,. . .,0IN)) *Variable*

space between last item in the level of points and the following text. Also, the space between the last item in the level of points and the next item in the previous level of points. For examples, see PTITEMGAP.

PTPOSTINDENT = structure (= (–1, –1, . . ., –1) *Variable*

If PTPOSTINDENT is –1 the first time PT is used at any level, the current indent value is stored, and TEXTFORM returns to that indent when EPT for that level is used.

PTPOSTLEVELSTRING = string structure (= ('','',. . .,'')) *Variable*

this string is included in the input before the PTPOSTGAP. See PTPRELEVELSTRING for an example.

PTPOSTSTRING = string structure (= ('. ','. ',') ',') ',') ','– ','– ','– ','– ','– ')) *Variable*

this string appears after the counter when lists of points are numbered automatically by PT. For examples, see PTCOUNTER and PTPOSTSTRING.

PTPREGAP = length structure (= (0IN,0IN,. . .,0IN)) *Variable*

is the space between preceding text and the first point at a particular level. For examples, see PTITEMGAP.

PTPREINDENT = structure (= (0,1,2,3,4,5,6,7,8,9)) *Variable*

is the indent of the indent before the PTPRESTRING. The number is the index into the LEFTINDENTS structure. To move all points over by two indents:

```
This is text before the points
<PTPREINDENT = PTPREINDENT+2>
<PTTEXTINDENT = PTTEXTINDENT+2>
<PT>An item at level 1
<PT>Another item at level 1
```

```
<PT(2)>An item at level 2
<PT>Another item at level 2
<PT(3)>An item at level 3
<PT(2)>Last item at level 2
<PT(1)>Last item at level 1
<EPT(1)>
```

produces:

```
This is text before the points
     1.    An item at level 1
     2.    Another item at level 1
           a.   An item at level 2
           b.   Another item at level 2
                1)   An item at level 3
           c.   Last item at level 2
     3.    Last item at level 1
```

PTPRELEVELSTRING = string structure (= ('','', . . .'')) *Variable*

this string is included in the input after the PTPREGAP. It can contain text or commands.

```
<PTPRELEVELSTRING(1) = '<L,SP ''*'',LEND>'
<PTPOSTLEVELSTRING(1) = PTPRELEVELSTRING(1)>
<PT>An item at level 1
<PT>Another item at level 1
<PT>Third item at level 1
<EPT>
```

produces:

```
****************************************************************
1.  An item at level 1
2.  Another item at level 1
3.  Third item at level 1
****************************************************************
```

PTPRESTRING = string structure (= ('','',. . .,'')) *Variable*

this string appears before the counter when lists of points are numbered automatically by PT. For examples, see PTCOUNTER.

PTRESTORE see PTSAVE *Macro*

PTSAVE *Macro*

Within points there are global variables which, once changed will remain changed for the remainder of the TEXTFORM run. These global variables are:

PTPREGAP
PTITEMGAP
PTPOSTGAP
PTPRELEVELSTRING
PTPREINDENT

PTPRESTRING
PTTEXTINDENT
PTTEXTINDENTHANG
PTPOSTLEVELSTRING
PTPOSTINDENT
PTPOSTSTRING
CTRn

PTSAVE, when issued before a list points, will save all of the global variables. Then you can make changes to a list of points, which remain in effect only until you issue PTRESTORE. The variables are then restored to their previous values.

PTSEP see OLDPT *Variable*

PTTEXTINDENT = structure (= (1,2,3,4,5,6,7,8,9,10)) *Variable*

This structure controls the indent command issued after the counter and PTPOSTSTRING. If you do not want the indent done for a level, set it to –1 for that level. For example, to replace the PTTEXTINDENT with your own indent command:

```
<pttextindent(1) = –1 >
<ptpoststring(1) = '<i l lindentindex+1, i h=2 l lindentindex–1>'>
This is text before the point.
<pt>This is a long test where points are several lines long. The
text around the counter is indented for two lines and then
returns to normal width. The text around the counter is indented
for two lines and then returns to normal width.
<pt>This is a long test where points are several lines long. The
text around the counter is indented for two lines and then
returns to normal width. The text around the counter is indented
for two lines and then returns to normal width.
```

produces:

```
This is text before the point.
1    This is a long test where points are several lines long. The
     text around the counter is indented for two lines and then
returns to normal width. The text around the counter is indented
for two lines and then returns to normal width.
2    This is a long test where points are several lines long. The
     text around the counter is indented for two lines and then
returns to normal width. The text around the counter is indented
for two lines and then returns to normal width.
```

PTTEXTINDENTHANG = structure (= (0,0,0,0,0,0,0,0,0,0)) *Variable*

controls whether the PTTEXTINDENT is done immediately, or on a later line. This produces hanging points, which are also the result of changing PTCOUNTER to zero, and changing PTPRESTRING and PTPOSTSTRING to null. PT(HANG) is the same as

```
<PTTEXTINDENTHANG(1) = 1>
```

```
<PTCOUNTER(1) = 0>
<PTPOSTSTRING(1) = ''>
```

QT *Layout Macro*

RC *System Variable 169*

REM(par1, par2) *Function 111*

REMAINING [(expression)] *Function 150*

expression

1
returns horizontal length measured from the current location to the bottom right-hand corner of the current footnote, float, reserve, or open text.

2
returns vertical length measured from the current location to the bottom right-hand corner of the current footnote, float, reserve, or open text.

REP(par1, par2) *Function 98*

RESERVE kwd [kwd2] [kwd3] [kwd4] name *Command 65*

kwd
is where the reserve appears on the page.

TOP
at the top of the page or column.

BOTTOM
at the bottom of the page or column. 'kwd4', specifying the size of the reserved area, must be given for bottom reserves.

kwd2
indicates if the reserve appears on front (right) or back (left) pages only. If this keyword is not specified, the reserve appears on both sides of the page.

FRONT or **R**IGHT

BACK or **L**EFT

kwd3

COLUMN
the reserve appears in each column of a logical page. The CURCOL system variable can be used to produce different results in different columns.

PAGE

kwd4

SIZE horizontal length vertical length
this specifies the size of the reserve. SIZE must fit within the *text* area of the page, i.e. inside margins. If SIZE is not given, a top reserve uses as much room as necessary. A bottom reserve *must* have a size specified.

horizontal length
can be a horizontal length, or the word DEFAULT. If DEFAULT is given, it is the same as PAGESIZE(1)–LEFTMARGIN–RIGHTMARGIN. If the horizontal length given is wider than the text area between left and right margins, but less than PAGESIZE(1), the reserve appears, but the left side of the reserve starts at the left margin. If the horizontal length given is wider than PAGESIZE(1), it produces the error *The RESERVE, or FLOAT is too wide. Maximum device width used.*

vertical length
If the size requested is larger than the text area on the page, the TEXTFORM run ends with: *RESERVES on page leave no room for text. Terminating.* If the size specified is too small to contain all the text, TEXTFORM prints the error message *Vertical size specified in RESERVE or FLOAT is too small.* If it is a bottom reserve, this message also appears: *The size of this page may be wrong.*

These errors can be produced if you specify a vertical size of 12 points for the reserve, and TEXTFORM then tries to start the reserve in the default LINESPACE, which is 12.0507 points.

name
may be the name of an item (command, variable, macro, or table) containing the text and/or commands. ‘name’ is evaluated when the reserve is printed. If ‘name’ is a macro, parameters can be given if the entire macro call is treated as a string, as in

```
<reserve top '&macrotop("Test")' >
```

However, the entire macro call is converted to upper case. To preserve lower case text:

```
<define &title = 'Test' >
<float top '&macrotop( &title )' >
```

All reserves need a name, so that they may be suspended. If the name is omitted, the following error appears: *All RESERVEs must have a 'name'. RESERVE ignored.*

RESERVE_INFO [(par, . . .)] *Function 67*

par

TOP

BOTTOM

FRONT

BACK

PAGE

COLUMN

SIZE

name of reserve

RES**ET**FOOT**NOTE**	*Command*	*57*
RIGHT	*Constant*	*48*
RIGHTGAP = length (=0)	*Keyword*	*77*
RIGHTINDENTS = length structure (= (.4IN,.8IN, . . ., 8IN))	*Variable*	*14*
RIGHTMARGIN = length (= 1IN)	*Variable*	*46*
RINDENT	*System Variable*	*18*
RINDENTINDEX	*System Variable*	*18*
ROMAN(par)	*Function*	*44*
RSFOOT see RESETFOOTNOTE	*Command*	*57*
RUNPAR	*Variable*	*168*

R0 = expression (= '') *Variable*

This variable TAKES and RETURNS values for functions which use register 0. The value in R0 will be converted to the required type before a call. If the function needs register 0 to be a pointer (PTR), then when the actual function call is made, register 0 will point to a copy of the contents of this variable (of the requested type).

R1 = expression (= '') *Variable*

This variable TAKES and RETURNS values for functions which use register 1. The value in R1 will be converted to the required type before a call. If the function needs register 1 to be a pointer (PTR), then when the actual function call is made, register 1 will point to a copy of the contents of this variable (of the requested type). See the description of the LOAD command for more details.

SCNTN (= a structure) *Constant*

a structure containing all of the special character names that are truly special characters.

S see SENTENCE *Command 155*

SENTENCE *Command 155*

SENTSEP = length (= 0.1IN) *Variable 155*

SEPARATE(par1, par2) *Function 107*

SERERRS = logical value (= TRUE) *Variable 170*

SHOWCPU *Function 169*

SHOWVM *Function 169*

SIZE *Layout Macro*

SOURCELNR *System Variable 169*

SP see SPLIT *Command 21*

SPLIT [kwd] [expression] *Command 21*

kwd

VERTICAL
If this keyword is used, ‘expression’ cannot be given.

expression (= SPLITSTRING)
‘expression’ may be null, or contain one or more characters, but commands are not evaluated.

SPLITSTRING = expression (= ' ') *Variable 21*

STACK(name) *Function 118*

STATS [(par)] *Function 176*

generates the statistics which appear after the source listing. If any ‘par’ is passed, the statistics are written on SERCOM, otherwise they are written in the source listing.

STOP *Command 168*

STRING(par) *Function*

STRUC(structure) *Function 106*

STRUCTURE(par1,par2, [par3]) *Function 106*

SUB *Layout Macro*

SUBSTRUC(par1,par2,par3) *Function 105*

SUS see SUSPEND *Command 65*

SUSPEND kwd *Command 65*

kwd
'kwd' is the facility to suspend:

RESERVE [kwd1] [name1]

kwd1
may be provided to specify a 'name' which appears on more than one reserve list. If 'kwd1' is omitted, 'name' will be removed from all reserve lists.

TOP
Search the RESERVE TOP list.

BOTTOM
Search the RESERVE BOTTOM list.

FRONT
Search the RESERVE FRONT list.

BACK
Search the RESERVE BACK list.

name1
if 'name1' appears, the list of names maintained by RESERVE is searched, and the name is removed if found. If 'name1' is not found, an error is generated: *No reserve by that name. SUSPEND command ignored.* If 'name1' is not supplied, all names on the list are removed.

AT kwd2 [name2]

kwd2
must be provided to specify which list 'name' will be on.

ASSIGN name3

END**O**F**C**OLUMN

END**OFF**ILE

END**O**F**L**INE

END**O**F**P**AGE

END**O**F**W**ORD

REFERENCE name3

START**O**F**C**OLUMN

TEXT**C**HARACTER c

name2
if 'name2' appears, the list of names maintained by AT is searched, and the name is removed if found. If 'name2' is not found, an error is generated. If 'name2' is not supplied, all names on the list are removed.

SYMMAX = expression (= 104976) *Variable 236*

SYMSIZE = expression (= 57344) *System Variable 236*

SYSCMD(par) *Function 169*

SYSCMDNOECHO(par) *Function 169*

T see TAB *Command 83*

TAB [expression [expression]] *Command 83*

TABLE_INFO(par1, par2 [,par3]) *Function 91*

par1
name of table or logicalpage

par2 (a string)

COLUMNS
returns the number of columns

WIDTH
returns the WIDTH of the column specified by 'par3'

GAP
returns the width of GAP for the column specified by 'par3'

LEFTGAP
returns the width of LEFTGAP

RIGHTGAP
returns the width of RIGHTGAP

ROWS
returns the number of rows (tables only)

HEIGHT
returns the vertical size of row n if 'par3' is specified, otherwise it returns the vertical size of the whole table (tables only)

ALIGNMENT
returns the ALIGNMENT of the column specified by 'par3' (tables only)

FONT
returns the FONT of the column specified by ‘par3’ (tables only)

par3
number of column required by some parameters as described above

TEXTONLY(par) *Function* *98*

TEXT_DESTINATION *Function* *150*

TEXTWIDTH(par) *Function* *101*

THEN see IF *Command* *156*

THISPAGE (= RIGHT) *System Variable* *48*

TIME *Constant* *149*

TIMERLIMIT = expression (= 1.0) *Variable* *181*

TITLE *Layout Macro*

TOC(par1 [,par2, par3, . . ., par10]) *Function* *129*

TOCHEADER = expression *Variable* *131*

'<PEND,ALIGNMENT=LEFT,I B 0,F 3>Table of Contents':-
'<F 1,LEND C, VG LINESPACE>':-
'<LEFTINDENTS=(.2IN,.4IN,.6IN,.8IN,. . .,1.8IN,2IN)>':-
'<RIGHTINDENTS=(.5in),SPLITSTRING=".">'

TOCHEADERLIST = structure (= (TOCHEADER, . . . ,TOCHEADER))
Variable *134*

TOCINDEX = expression (= 1) *Variable* *134*

expression
a value from 0 to 99

TOCMACRO *Macro* *133*

The contents of the macro are:
<LEND, I B PAR(1) 1, I H=1 L PAR(1)+1,KE,-
PAR(3), . . ., PAR(10), I R OFF, " ", SP,-
PAR(2), EKE, LEND>

TOCMACROLIST = structure (= ('TOCMACRO', . . . ,'TOCMACRO'))
Variable *134*

TOP *Constant* *53*

TOOD expression — *Command*

expression
hexadecimal string

TOPD expression — *Command*

expression
string

TOPMARGIN = length (= 1IN) — *Variable* *46*

TRANSLATE kwd character1 character2 — *Command*

This command is de-emphasized. Use AT TEXTCHARACTER instead.

TRUE (= 1) — *Constant* *102*

TSIZE (= 0.1389IN) — *System Variable* *43*

TYP see TYPESIZE — *Command* *43*

TYPE(name) — *Function* *122*

TYPESIZE length (0.1389IN) — *Command* *43*

U see UNDERLINE — *Command* *37*

UC (= '<CAP=TRUE>') — *Variable*

UERR(par1 [,par2]) — *Function* *171*

UN — *Constant*

UNDERLINE [expression] — *Command* *37*

expression

OFF

ON

UNDERLINEDISPLACEMENT = length (= 0) — *Variable* *37*

UNDERLINESTRING = expression (= '_') — *Variable* *37*

UNDERLINEWORDSPACE = logical value (= FALSE) — *Variable* *37*

UNFORMATTED — *Constant* *93*

UNIT — *Constant*

UNITS *Constant*

UNSTACK(name) *Function* 118

UPPERCASE(par) *Function* 44

UPPERCASEINPUT = logical value (= FALSE) *Variable*

If this varible is TRUE, all input will be translated to lower case before being processed by TEXTFORM. Translation does not begin until the next line.

USE [name] (PHYSICALPAGE) *Command* 74

VECLEN(name) *Function* 105

VERSION (= string) (= 1.50) *Constant* 170

VERTALIGNMENT = kwd *Variable* 53

kwd

TOP

BOTTOM

CENTRE or CENTER

BOTH

JUSTIFY

VERT**G**AP length *Command* 52

VERT**J**UST *Command* 54

VERT**S**PACE length *Command* 51

VG see VERTGAP *Command* 52

VJ see VERTJUST *Command* 54

VS see VERTSPACE *Command* 51

VTYPE(name) *Function* 105

WHILE 'comparison' DO name *Command* 159

WIDTH = length *Keyword* 91

WORDSPACE = length (= 0.1IN) *Variable* 18

X(par1 [,par2, par3, . . ., par10]) *Function 135*

XCOUNT = expression *Variable 138*

XHEADER = expression *Variable 137*

'<PEND,ALIGNMENT=LEFT,I B 0,F 3>Index':–
'<F 1,LEND C,VG LINESPACE>':–
'<LEFTINDENTS=(.2IN,.4IN,.6IN,.8IN,. . .,1.8IN,2IN)>':–
'<RIGHTINDENTS=(.5IN)>'

XHEADERLIST = structure (= ('XHEADER', . . ., 'XHEADER')) *Variable 138*

XINDEX = expression (= 1) *Variable 138*

expression
a value from 0 to 99

XMACRO *Macro 138*

The contents of the macro are:
<LEND,I B PAR(1) 0,I H=1 L PAR(1)+1, –
XPARS=NRPARS-1,PAR(3), –
com print entries, FOR 'XCOUNT=4' UNTIL 'XPARS' DO XPRINT, –
XPARS=PAR(2), –
com print pagenos, FOR 'XCOUNT=1' UNTIL 'XPARS' DO
XPGPRINT>

XMACROLIST = structure (= ('XMACRO', . . . , 'XMACRO')) *Variable 138*

XPARS = expression *Variable 138*

XPGPRINT = expression (= '<'', '',XPAGENUM(XCOUNT)>') *Variable 138*

XPRINT = expression (= '<'', '',PAR(XCOUNT)>') *Variable 138*

YEAR *Constant 149*

YES (= 1) *Constant 102*

APPENDIX 2 – TEXTFORM'S INTERNAL STORAGE

In the TEXTFORM program, the **symbol table** contains every item—predefined or user-defined—used in the TEXTFORM language, and its value and attributes (variable, command, macro, special character, function). Any item encountered within command mode must be a recognized item in the symbol table. Specifically, a command causes an immediate formatting action or change to the symbol table. Alternately, any string which appears in command mode will appear in the text.

Predefined items are defined by TEXTFORM, and are in the symbol table at the beginning of a TEXTFORM run. A predefined item may not be erased. User-defined items are added to the symbol table via the DEFINE command, and may be changed or erased.

In the symbol table, **garbage space** may be created when the user (or a layout macro, or TEXTFORM itself) changes the value of a variable, or erases something, leaving extra storage space available. For example, if a variable (such as a string or structure) becomes longer, and it was not the last variable defined, it must be moved in storage. This results in garbage space. If an item is erased, and it was not the last item defined, this also results in garbage space.

GARBAGESPACE contains the number of bytes of waste space in the symbol table. **Garbage collection** occurs when the GARBAGESPACE exceeds 25% of the symbol table size (SYMSIZE). If symbol table size exceeds the maximum allowed (SYMMAX), a garbage collection is performed, and SYMMAX is doubled. GARBAGECOUNT contains the number of garbage collections of the symbol table during the run.

Garbage collection is accomplished by getting enough space to hold the current contents (SYMSIZE–GARBAGESPACE), then copying the current contents of the symbol table to the new area, and releasing the old. Garbage collection will generally not occur unless the document is large (over 300 pages), a large number of items are created and erased, or one or more items are continually getting larger. Documents in these categories can benefit from changing SYMMAX, or by careful definition and setting of 'highly mobile' variables. For example, if a structure will eventually have 100 elements, added one by one, indicate this when you define the structure name, as in:

```
<define  &structure,  &structure(100)  =  ' '  >
```

INDEX

Bold page numbers indicate where the word is defined.

abbreviation, of TEXTFORM
  commands, 32
ABORT command, 168, 184
addition, addition (+) operator, 110
alignment, 11
  and indent changes, 17
  in ASIS or PREFORMATTED
    mode, 94
  left right centre or both, 203
ALIGNMENT keyword, 92
ALIGNMENT variable, 11, 184
ALLOWLINEBREAK command, 22, 184
alphabetic, counter, 186
  integer displayed as, 44, 186
alphabetic counters, 124
ALPHABETIC function, 44, 184
APL characters, 40, 205
arabic numerals, 124, 186
as is mode, 93
ASIS constant, 94, 184
assign values to variables, 69, 96
asynchronous, **69**
AT, AT-points, 69, 151, 184, 230
  suspending AT-points, 230
AT command, 42, 69, 132, 184
AT_INFO function, 185
attention interrupt, 180
ATTNS system variable, 180, 185
ATTRIBUTE command, 123, 185
attribute of variables, 112, 123, 131,
  209, 221
A0 – A8 constant, 187
BACK constant, 187
back page, 198, 226
backspace, 41
balanced columns, 88
baseline, **43**
begin, column, 78
  line, 6
  page, 8
  paragraph, 8
bibliography, 96, 139
  formatting, 9
blank space, 19, 20, 21
  blank pages, 60, 152
  character, 11
  recommended before endfootnote
    command, 56
  when allowed in a command, 13
  when not allowed at end of input
    line, 27
  when not allowed in input, 94
  when not allowed in interactive
    input, 179
BLANKCHARACTER command, 20, 42,
  154, 187
BLANKLINE variable, 187
body, in TEXTFORM notation, **32**
BOLD constant, 187
BOLDITALIC constant, 187
BOTH constant, 187
BOTMARGIN keyword, 77
BOTMARGIN variable, 46, 151, 187
bottom, column, 197, 226
  page, 197, 226
BOTTOM constant, 187
boxes, 144
BREAK GCI command, 179
CalComp, 181
CAP variable, 187
capital letters, in commands, 3
  in TEXTFORM notation, 32
CAPNEXTCHAR (@) constant, 30, 188
CASE command, 158, 188
catenation, catenation ($) operator, **111**
  catenation (:) operator, 28
  implicit, 167
  structure, 106
CENTER constant, 188
CENTRE constant, 188
centred text, 11, 12
character, 11, **25**, 30, 107, 154
  . . ., 32
  <, 31
  +, 108, 110
  |, 99
  &, 34
  $, 108
  *, 111
  ;, 99
  /, 111
  _, 38
  :, 111
  #, 102

@, 30, 102, 108
', 97
", 97
=, 27
[, 32
replacing automatically, 42
character set, **25**, 39, 205
CHARACTERSET command, 39, 188
CHAREXIST function, 40, 188
CHARS function, 98, 188
CHHRZ command, 188
CHPSZ command, 188
CHVRT command, 188
column, 152, 198, 226
balanced columns, 78, 87, 88
begin, 78
column width, 77, 91, 150
defining a specific, 91
defining columns, 192
defining columns in tables, 87
defining columns on logical page, 76
multiple columns on page, 76
number of the current, 81
COLUMN command, 78, 188
COLUMN_POSITION function, 150, 153, 189
COLUMNEND command, 53, 78, 189
COLUMNS keyword, 189
in logical page, 76, 193
COMDINT (<) constant, 31, 189
COMDSEP (,) constant, 189
COMDTERM (>) constant, 31, 189
command, **25**, 27
command initiator and terminator, 26, 31
command mode, 119, 167
command separator, 26, 156
COMMENT command, 26, 189
comparison, 156, 159
conditional control, 120, 149
AT, 69
CASE, 158
FOR, 159, 181
IF, 156
WHILE, 159, 181
constant, **123**
TEXTFORM, 149
contents, 129
appearance of table of contents entries, 129
contents page, 131
creating contents entries, 129
macro to create entries in table of contents, 115
multiple tables of content, 134
TOC (contents) function, 129
continuation, 27, 28, 31, 115
CONTINUE GCI command, 179
CONTLINECHAR (-) constant, 31, 189
cross reference, 176, 209
sample, 176, 177
CROSSREFERENCE command, 167, 176, 189
CTRn variable, 190
CURBOTMARGIN system variable, 47, 151, 191
CURCOL system variable, 81, 191
CURCS system variable, 39, 191
CUREMPFONT function, 38, 191
CURFONT system variable, 38, 149, 191
CURKEYBOARD system variable, 191, 206
CURLEFTMARGIN system variable, 47, 151, 191
CURLINESPACE system variable, 153, 191
CURLP system variable, 74, 191
CURPAGEHEIGHT system variable, 151, 191
CURPAGESIZE system variable, 46, 151, 191
CURPAGEWIDTH system variable, 151, 191
CURRIGHTMARGIN system variable, 47, 151, 191
CURTABLINE system variable, 90, 191
CURTOPMARGIN system variable, 47, 151, 191
CURUNDERLINE system variable, 37, 191
DATE constant, 149, 191
DAY constant, 149, 191
default, **5**, 55
keyword, 198
default state, **55**
default units, **31**
DEFCOLGAP keyword, 76, 77, 191
DEFCOLWIDTH keyword, 76, 77, 191
define, **34**
column in table or logical page, 91
logical page, 75
macro, 113
table, 82
variable, 95
DEFINE COLUMN command, 76, 91, 192
DEFINE command, 34, 95, 191
DEFINE LOGICALPAGE command, 75,

192
DEFINE MACRO command, 113, 193
DEFINE TABLE command, 82, 193
DEFUNITS variable, 31, 194
DELAY command, 72, 194
delimiter, **27**
string delimiters, 97, 114
delimiter (') character, 27
device independent, **25**
dictionary-style page titles, 66
DISABLE MESSAGE command, 170, 194
discretionary hyphen, **22**
DISPLAY command, 92, 171, 194
DISPLAY GCI command, 179
DIV function, 111, 195
division, division (/) operator, 111
integer, 111
remainder of, 111
DO keyword, 159, 200
document, compose a document, 1
double space, 15
DRAW command, 145, 195
dynamic, **222**
element in structure, **103**
ELSE command, 156, 202
emphasis, 10
emphasis character, 30
font, 39
EMPHNEXTCHAR (_) constant, 38, 195
ENABLE MESSAGE command, 170, 195
end, column, 69, 78
line, 69
page, 69
sentence, 19, 155
word, 69, 71, 154
end of file, 69, 70, 132, 168
ENDDEFINE command, 191
column, 91
logical page, 75
macro, 113
table, 82
ENDFLOAT command, 197
ENDFOOTNOTE command, 56, 200
ENDIF command, 156, 202
ENDKEEP command, 50, 205
English, hyphenation, 23
integer displayed as, 44
ENGLISH function, 44, 196
EPT macro, 9, 196
erase, 123
macro, 116
name, 116, 122
ERASE command, 116, 122, 197
error, 169
error messages, 170
error messages in listing, 175
generating your own error messages, 170
how to interpret error messages, 3
how to stop expected error messages from printing, 170
ERROR command, 117, 169, 170, 197
ERROR GCI command, 179
execute, 119
execute ($) operator, **108**
EXIST function, 122, 197
expression, **112**
in command, 32, 109
in TEXTFORM notation, 32
EXTRABOLD constant, 197
EXTRABOLDITALIC constant, 197
FALSE constant, 197
figure, figure captions, 61
file editor, 1
files related to TEXTFORM, TXTF:TXTF.MACROS, 58
files used by TEXTFORM, –TXTFLIST, 172, 209
–TXTFTC01, 132
–TXTFXI01, 137
–TXTFXM01, 142
–TXTFXR1, 176
–TXTFXR2, 176
–TXTFXS01, 137
*TXTFHYPHDICT, 23
ETC:TXTFHYPHDICT, 24
TXTF.PROLOG, 167
float, appearance of floated text, 60
bottom, 61
float table or figure space, 61
top, 60
very top or bottom, 61
FLOAT command, 60, 197
FLOAT_INFO function, 199
FOGS command, 199
font, **26**, 38
bold font, 38
current, 39
emphasis font, 38
italic font, 10, 38, 130
FONT command, 38, 200
FONT keyword, 92
FOOTCONTINUE variable, 57, 200
FOOTCTR variable, 57, 70, 200
FOOTDIVIDE variable, 57, 200
FOOTINDEX variable, 57, 200
footnote, 56
at end of chapter, 58

continued on next page, 57
counter, 57, 70
in tables, 85
on logical page, 79
separated from text, 9, 57
FOOTNOTE command, 56, 200
FOOTSEP variable, 57, 200
FOR command, 159, 200
FORDIS function, 131, 200
format, **25**
formatting actions taken by TEXTFORM, 151
French, hyphenation, 23
integer displayed as, 44
FRENCH function, 44, 200
FRONT constant, 200
front page, 198, 226
function, **160**
FUNCTIONRC system variable, 160, 200
FUZZ variable, 157, 200
GALLEY variable, 200, 216
GAP keyword, 91, 201
garbage collection, **236**
garbage space, **236**
GARBAGECOUNT system variable, 201, 236
GARBAGESPACE system variable, 201, 236
GCI command, 177, 236
global variable, **120**
GUSER I/O unit, 167
HANG keyword, 16
hanging indent, **16**
hexadecimal, 102
HEXDELIM (#) constant, 201
highlight, 38
horizontal space, 18, 21, 229
HORSPACE command, 19, 154, 201
hyphenation, 22, 209
algorithm, 23
dictionary, 23
discretionary hyphen, 22
internal dictionary, 201
HYPHENATION command, 23, 201
I/O unit, 163, 167
IF command, 156, 202
IN constant, 202
INCH constant, 202
INCHES constant, 202
INCLUDE command, 63, 202
indent, **13**
current number, 18
delayed, 16
hanging, **16**
in tables, 85
left and right, 13, 153
of paragraph, 8
paragraph indent, 153
size, 18
INDENT command, 13, 203
index, 135
appearance of index, 137
creating index entries, 135
index page, 137
maximum length of entries, 138, 142
multiple indexes, 138
number of levels of indexing, 138, 142
of a structure element, **103**
sorting sequence for index, 142
TIMERLIMIT while printing, 139
X (index) function, 135
INDEX command, 141, 204
input, **1**, 25
blanks in input, 2, 11
input line length, 2
line number, 169
maximum number of characters in input line, 167
preparing input, 2
uppercase or lowercase or mixed case, 30
INPUTMODE variable, 93, 119, 204
INSERT GCI command, 179
inside margin, **49**
INTERACTIVE command, 177, 204
INTPAGELNR command, 205
INTPAGELNR system variable, 173
INTPAGENR command, 205
INTPAGENR system variable, 173
ITALIC constant, 205
italic text, 10, 38
ITYPE function, 127, 205
justified column, 53
justified text, **11**, 12, 19, 29
justify, **11**
JUSTIFY constant, 205
KEEP command, 50, 205
keep text together, 50
keep words on same line, 29
KEYBOARD command, 40, 205
keyword, as executed variable, 109
in a command, **13**, 32
in TEXTFORM notation, 32
preceded by ¬ or –, 32
kwd, in TEXTFORM notation, 32
LAYNAME system variable, 206
layout (prepared formats), and logical

pages, 81
LAYOUT command, 206
LC variable, 207
left alignment, 11, 12, 203
LEFT constant, 207
left indent, 13, 203
left margin, current, **203**
LEFTGAP keyword, 77, 87, 207
LEFTINDENTS variable, 13, 203, 207
LEFTMARGIN keyword, 77
LEFTMARGIN variable, 46, 151, 207
length, in TEXTFORM notation, **33**
length of column, 203
negative lengths, 100
units of measurement, 31, 100
letter, form letter, 116
letter space, 12
LI constant, 207
LIGHT constant, 207
LIGHTITALIC constant, 207
LINDENT system variable, 149, 203, 207
LINDENTINDEX system variable, 149, 203, 207
line, at top of page, 7
beginning a line, 5
end of line, 153
unit of measurement, **101**
LINE command, 5, 207
line drawing, 144
appearence of line, 145
information of line position, 147
position of line, 144
LINEEND command, 5, 12, 207
LINES constant, 207
linespace, single space, 15
LINESPACE variable, 5, 207
list, enumerated list, 9, 44, 196, 215, 219
LIST command, 172, 208
and contents of macro, 119
and par= in run command, 167
display of table or logical page, 92
example of use, 175
listing, 172, 208
inserting text into, 172
load, **160**
LOAD command, 160, 210
loaded, **35**
LOCAL command, 120, 213
local variable, 120
LOCATION command, 144, 213
LOCATION_INFO function, 147, 213
logical page, **74**, 152
and layouts, 81
defining logical page, 75
information about dimensions, 76, 92
name of current, 74
page number on logical page, 81
reserve on logical page, 81
tables on, 90
using logical page, 74, 75, 78
logical value, 102, 156
in TEXTFORM notation, **33**
logical variable, 102
LOGICALBACKSPACE command, 41, 154, 213
lower case, 44, 126
LOWERCASE function, 44, 213
MACFLAG system variable, 118, 213
macro, 109, 113, 118, 208
changing a macro, 116
index entries, 136
parameters, 113
MACWRITE function, 162
margin, 25, 46
MAX function, 106, 213
MAXWORDSPACE variable, 12, 214
measurement, 100
MEMBER function, 105, 214
meta-character, 29
disabling, 42
MILLIMETER constant, 214
MILLIMETERS constant, 214
MILLIMETRE constant, 214
MILLIMETRES constant, 214
MIN function, 106, 214
MINHYPH variable, 24, 214
MM constant, 214
monospace, **26**
MONTH constant, 149, 214
MTS command, CONTINUE with, 167
CREATE, 1
EDIT, 1
RESTART, 169
RUN, 2, 167
SIGNON, 1
MTS GCI command, 179
MTSCMD function, 132, 169, 214
multiplication, multiplication (*) operator, 111
name, 34, 122
in TEXTFORM notation, **33**
NEWLINE command, 214
NEWPARA command, 8, 215
NO constant, 215
non-linebreaking word space, 29
NONLBWDSPACE (¬sign) constant, 29, 215

NOPROLOGFILE command, 167, 215
NORMAL constant, 215
NOSENTENCE command, 155, 215
NRPARS function, 117, 215
null parameter, 114
null string, **95**, 114
number, 99
    in TEXTFORM notation, 33
    scaled, 99
numbering, 124, 219
ODCHARACTERSET variable, 39, 215
ODLOADED system variable, 149, 215, 216
ODNAME variable, 215
OFF constant, 215
OLDPT macro, 215
ON constant, 216
open text, **53**, **54**
operator, 99, 102, 107
orphan, 50
output device, 25, 34, 216
    page printer, 48
output line, **1**
OUTPUTDEVICE command, 34, 216
page, **25**, 151, 198
    blank page, 47, 60
    end of page, 3, 151
    front and back (left and right), 48
    logical page, 74
    page dimensions, 46, 90, 150, 151, 187
    physical page, 74
PAGE command, 8, 47, 217
page number, 48, 49, 217
    in table of contents entries, 129, 130
    on logical page, 81
page printer, 206
page processor, **151**
PAGE_POSITION function, 150, 151, 217
PAGEEND command, 47, 217
PAGENUM constant, 217
PAGESIZE keyword, 77
PAGESIZE variable, 46, 151, 217
PAR command, 113, 217
par field, 167
PARAEND command, 217
paragraph, 8, 71
PARAIND variable, 8, 217
parameter, **28**, 33
    function, 160
    macro parameters, **113**, 160
    null parameter, 114
    number of parameters passed to macro, 117
PARASEP variable, 8, 217
PDCHARACTERSET variable, 218, 219
PDLOADED system variable, 218, 219
PDNAME variable, 218, 219
PGNTLIMIT variable, 183, 218
PGNTS system variable, 183, 218
physical page, **74**
PHYSICALPAGE, 74
PI constant, 218
PICA constant, 218
PICAS constant, 218
PNCTR variable, 81, 130, 149, 151, 217, 218
PNHEAD variable, 217, 218
PNTRAIL variable, 217, 218
PO constant, 218
point, unit of length, **43**
POINT constant, 218
pointer, 102
    pointer (@) operator, 108
POINTS constant, 218
POSITION function, 150, 154, 218
predefined variable, **95**
PREFORMATTED constant, 93, 218
print (*PRINT*), 3
printed output, 3, 168
printer, 25, 206
PRINTON variable, 48, 151, 218
program interrupt, 182
prolog file, 167
proof, 172, 219
    proof device, 35, 172, 219
    sample of a proof, 36, 173
PROOF variable, 219
PROOFDEVICE command, 35, 172, 219
proportional, **26**
PT macro, 9, 219
    CTRn variables, 221
    PT... variables, 196, 219, 221
queue, **119**
RC system variable, 169, 171, 226
readability measurement, 199
recursion, **120**
references, 58
REM function, 111, 226
REMAINING function, 118, 150, 152, 226
REP function, 98
replacement character, 40
reserve, 49, 152, 230
RESERVE command, 63, 226
RESERVE_INFO function, 227
RESETFOOTNOTE command, 57, 228
return code, **160**, 171

right alignment, 11, 12, 203
RIGHT constant, 228
right margin, current, **203**
RIGHTGAP keyword, 77, 228
RIGHTINDENTS command, 14
RIGHTINDENTS variable, 203, 228
RIGHTMARGIN keyword, 77
RIGHTMARGIN variable, 46, 151, 228
RINDENT system variable, 149, 203, 207
RINDENTINDEX system variable, 149, 203, 207
roman, character, 26
    integer displayed as, 44
ROMAN function, 44, 228
run, errors which end the run, 169
    restarting the run, 169
    TEXTFORM run, 2, 129, 167
RUN GCI command, 179
RUNPAR variable, 168, 228
R0 variable, 228
R1 variable, 161, 228
sample documents formatted with TEXTFORM, form letter, 116
save values of variables, 118
SCARDS I/O unit, 167
SCNTN constant, 228
sentence, 19, 155
SENTENCE command, 229
SENTSEP variable, 19, 155, 229
SEPARATE function, 107, 229
SERCOM I/O unit, 4, 92, 167, 170
SERERRS variable, 170, 229
SHOWCPU function, 169
SHOWVM function, 169
single space, 15
size of characters, 43
SKIP GCI command, 179
source file, **25**
source line, 173
SOURCELNR system variable, 169, 229
space, between columns, 77
    between letters, 12
    between words, 12
    between words in a macro, 116
space (blank space character), 20, 154
    underlining, 20, 37
space (horizontal space), 19, 150
    for float, 198
    from preceding column, 91
    remaining on line, 150
    split, 21, 229
space (vertical space), 53, 61, 150
    at top of page or column, 51
    between footnotes, 57
    between rows in table, 84
    for float, 198
    on the column, 51
    preceding paragraph, 8
    remaining on column, 150
    split vertical, 229
    VERTGAP, 52
special character, 29, 40, 94, 205, 228
SPLIT command, 21, 52, 154, 229
SPLITSTRING variable, 21, 22, 229
SPRINT I/O unit, 35, 167, 172, 176
SPUNCH I/O unit, 35, 167, 200
STACK function, 118, 229
start, column, 47, **152**
    line, 7, 153
    page, 8, **151**
    paragraph, 8
statistics, 175, 176, 209, 229
    about readability, 199
STATS function, 229
STEP GCI command, 179
STEP keyword, 159, 200
STOP command, 168, 229
STOP GCI command, 179
string, **27**, 33, 130, 229
    doubling meta-character in, 163
    null, **95**
    number of characters in, 98, 126
    substring from-length (|) operator, 98
    substring from-to (;) operator, 98
    variables, 97
STRING function, 229
STRUC function, 106, 229
structure, **103**, 124, 127, 157
    catenation, 106
    defining a structure, 103
    efficient use of large, 236
    finding index of an element, 105
    finding maximum or minimum element, 106
    finding number of elements, 105
    in TEXTFORM notation, 33
    index of structure, **103**
    substructure, 105
STRUCTURE function, 106, 229
subroutine, 160
    MTS system subroutine, 160, 210
    TEXTFORM called as, 163
    user-written, 161
subscript, **40**, 162
substring, **98**, 118
SUBSTRUC function, 105
superscript, **40**, 57, 162
suspend, at points, 71

reserves, 65
SUSPEND command, 65, 71, 230
symbol table, 197, 209, 234, **236**
SYMMAX variable, 231, 236
SYMSIZE system variable, 231, 236
synchronous, **69**
SYSCMD function, 169, 231
SYSCMDNOECHO function, 169, 231
system variable, **95**, 118, 149
TAB command, 83, 231
table, 61, 82, 150
changing an entry in a table, 89
creating a table, 82, 89
emptying a table, 82
entry, 83
filling a table, 82, 83
in reserve, 89
indents in, 85
information about dimensions, 91
placing footnote in, 85
printing a table, 82
reusing a table, 83
using a table, 83
using a table on logical page, 90
TABLE_INFO function, 91, 231
text, **25**
bold italic normal, 38
emphasis, 10
kept together, 29, 60
removing commands from, 98
reproduced as entered, 93, 119, 154
size, 43
special positioning, 153
text mode, 29, 119
text processing, **25**
TEXT_DESTINATION function, 150, 232
TEXTONLY function, 98, 232
TEXTWIDTH function, 18, 20, 101, 154, 232
THEN command, 156, 202
THISPAGE system variable, 48, 149, 151, 232
TIME constant, 149, 232
timer interrupt, 181, 232
TIMERLIMIT variable, 139, 181, 232
TOC function, 129, 232
TOCHEADER variable, 131, 133, 232
TOCHEADERLIST variable, 134, 232
TOCINDEX variable, 134, 232
TOCMACRO macro, 133, 232
TOCMACROLIST variable, 134, 232
TOOD command, 233
top, column, 197, 226
page, 197, 226
TOP constant, 232
TOPD command, 233
TOPMARGIN keyword, 77
TOPMARGIN variable, 46, 151, 233
TRUE (#00#) constant, 233
TSIZE system variable, 43, 233
TXTF.PROLOG, 167
TYPE function, 122, 233
typesetting, 43, 101
typesize, 43
current typesize, 43
TYPESIZE command, 43, 233
UC variable, 233
UERR function, 171, 233
UN constant, 233
underline, 37
how to underline blanks, 37
underline character, 38
UNDERLINE command, 37, 233
UNDERLINEDISPLACEMENT variable, 37, 233
UNDERLINESTRING variable, 37, 233
UNDERLINEWORDSPACE variable, 37, 233
UNFORMATTED constant, 93, 233
unit, unit of measurement, 100
UNIT constant, 233
unit of measurement, **100**
UNITS constant, 233
UNSTACK function, 118, 234
UNTIL keyword, 159, 200
upper case, 44, 125, 187, 234
commands, 3
forcing characters to be, 30
UPPERCASE function, 44, 234
UPPERCASEINPUT variable, 234
USE command, 74, 75, 78, 83, 84, 234
value, 124, 209
maximum value, 125
minimum value, 125
variable, **27**, 95, 159
defining a variable, 34, 95
displaying a variable, 112, 124, 131
executing a variable, 108, 119
global variable, **120**
local variable, 120
predefined variable, **95**
save and restore value of variable, 118
system variable, **95**, 149
user-defined variable, 95
VECLEN function, 105, 234
VERSION constant, 170, 234
VERTALIGNMENT variable, 53, 234
VERTGAP command, 52, 234

vertical space, 51, 52, 53, 61, 229
    in tables, 84
VERTJUST command, 54, 234
VERTSPACE command, 7, 234
VTYPE function, 105, 234
WHILE command, 159, 234
widow, **50**
    preventing via macro, 117
width, width of a character, 18, 26
    width of column, 77, 91
    width of text, 101
WIDTH keyword, 91, 234
word, **11**, 26
    current word, 153
    words don't fit on output line, 22, 26
word space, 18, 154
WORDSPACE variable, 18, 154, 234
X function, 135, 235
XCOUNT variable, 138, 235
XHEADER variable, 137, 235
XHEADERLIST variable, 138, 235
XINDEX variable, 138, 235
XMACRO macro, 138, 235
XMACROLIST variable, 138, 235
XPARS variable, 138, 235
XPGPRINT variable, 138, 235
XPRINT variable, 235
YEAR constant, 149, 235
YES constant, 235

INDEX TO TEXTFORM LANGUAGE

command, ABORT, 184
- ALB, 184
- **A**LLOW**L**INE**B**REAK, 184
- AT, 184
- ATT, 185
- **ATT**RIBUTE, 185
- BC, 187
- **B**LANK**C**HARACTER, 187
- C, 187
- CASE, 188
- CEND, 188
- **C**HARACTER**S**ET, 188
- CHHRZ, 188
- CHPSZ, 188
- CHVRT, 188
- **C**OLUMN, 188
- **C**OLUMN**END**, 189
- **COM**MENT, 189
- CR, 189
- **C**ROSS**R**EFERENCE, 189
- CS, 190
- D, 191
- DEF, 191
- **DEF**INE, 191
- **DEF**INE [kwd] **COL**UMN, 192
- **DEF**INE [kwd] **L**OGICAL**P**AGE name, keywords,, 192
- **DEF**INE [kwd] **MA**CRO, 193
- **DEF**INE [kwd] **TA**BLE, 193
- DELAY, 194
- DISABLE, 194
- **D**ISPLAY, 194
- DRAW, 195
- EDEF, 195
- EFL, 195
- EFOOT, 195
- EIF, 195
- EKE, 195
- ELSE, 195
- ENABLE, 195
- ENDDEF, 195
- ENDDEFINE, 195
- **E**ND**FL**OAT, 196
- **E**ND**FOOT**NOTE, 196
- ENDIF, 196
- **E**ND**KE**EP, 196
- ER, 197
- **ER**ASE, 197
- ERR, 197
- **ERR**OR, 197
- F, 197
- FL, 197
- **FL**OAT, 197
- FOGS, 199
- **F**ONT, 200
- FOOT, 200
- **FOOT**NOTE, 200
- FOR, 200
- GCI, 201
- **H**OR**S**PACE, 201
- HS, 201
- **HYPHEN**ATION, 201
- I, 202
- IF, 202
- INC, 202
- **INC**LUDE, 202
- **I**NDENT, 203
- INDEX, 204
- INT, 204
- **INT**ERACTIVE, 204
- K, 205
- **KE**EP, 205
- **K**EYBOARD, 205
- L, 206
- **LA**YOUT, 206
- LBS, 206
- LEND, 207
- **L**INE, 207
- **L**INE**END**, 207
- LIST, 208
- LOAD, 210
- **LOC**AL, 213
- LOCATION, 213
- **L**OGICAL**B**ACK**S**PACE, 213
- MTS, 214
- NC, 214
- **N**EW**C**OL, 214
- **N**EW**L**INE, 214
- **N**EW**PAGE**, 215
- **N**EW**P**ARA, 215
- NL, 215
- **NOPR**OLOGFILE, 215
- **N**O**S**ENTENCE, 215
- NP, 215
- NPAGE, 215
- NS, 215
- OD, 215
- **O**UTPUT**D**EVICE, 216

OVERSTRIKE, 216
P, 217
PAGE, 217
PAGE**END**, 217
PAR, 217
PARA**E**ND, 217
PD, 217
PE, 218
PEND, 218
PROOF**D**EVICE, 219
RESERVE, 226
RES**ET**FOOTNOTE, 228
RSFOOT, 228
S, 229
SENTENCE, 229
SP, 229
SPLIT, 229
STOP, 229
SUS, 230
SUSPEND, 230
T, 231
TAB, 231
THEN, 232
TOOD, 233
TOPD, 233
TRANSLATE, 233
TYP, 233
TYPESIZE, 233
U, 233
UNDERLINE, 233
USE, 234
VERT**G**AP, 234
VERT**J**UST, 234
VERT**S**PACE, 234
VG, 234
VJ, 234
VS, 234
WHILE, 234
constant, ASIS, 184
A0 – A8, 187
BACK, 187
BOLD, 187
BOLDITALIC, 187
BOTH, 187
BOTTOM, 187
CAPNEXTCHAR, 188
CENTER, 188
CENTRE, 188
COMDINT, 189
COMDSEP, 189
COMDTERM, 189
CONTLINECHAR, 189
DATE, 191
DAY, 191
EMPHNEXTCHAR, 195
EXTRABOLD, 197
EXTRABOLDITALIC, 197
FALSE, 197
FRONT, 200
HEXDELIM, 201
IN, 202
INCH, 202
INCHES, 202
ITALIC, 205
JUSTIFY, 205
LEFT, 207
LI, 207
LIGHT, 207
LIGHTITALIC, 207
LINES, 207
MILLIMETER, 214
MILLIMETERS, 214
MILLIMETRE, 214
MILLIMETRES, 214
MM, 214
MONTH, 214
NO, 215
NONLBWDSPACE, 215
NORMAL, 215
OFF, 215
ON, 216
PAGENUM, 217
PI, 218
PICA, 218
PICAS, 218
PO, 218
POINT, 218
POINTS, 218
PREFORMATTED, 218
RIGHT, 228
SCNTN, 228
TIME, 232
TOP, 232
TRUE, 233
UN, 233
UNFORMATTED, 233
UNIT, 233
UNITS, 234
VERSION, 234
YEAR, 235
YES, 235
function, ALPHABETIC(par), 184
AT_INFO(par1 [,par2]), 185
CHAREXIST(par), 188
CHARS(par), 188
COLUMN_POSITION
[(expression)], 189
CUREMPFONT, 191
DIV(par1, par2), 195
ENGLISH(par), 196

EXIST(name), 197
FLOAT_INFO [(par...)], 199
FORDIS(name), 200
FRENCH(par), 200
ITYPE(name, [par2]), 205
LINEREM, 207
LINEUSED, 207
LISTING(par), 210
LOCATION_INFO(par), 213
LOWERCASE(par), 213
MAX(par1,par2), 213
MEMBER(par,structure), 214
MIN(par1,par2), 214
MTSCMD(par), 214
NRPARS, 215
PAGE_POSITION [(expression)], 217
PAGEREM, 217
PAGEUSED, 217
POSITION [(expression)], 218
REM(par1, par2), 226
REMAINING [(expression)], 226
REP(par1, par2), 226
RESERVE_INFO [(par, ...)], 227
ROMAN(par), 228
SEPARATE(par1, par2), 229
SHOWCPU, 229
SHOWVM, 229
STACK(name), 229
STATS [(par)], 229
STRING(par), 229
STRUC(structure), 229
STRUCTURE(par1,par2, [par3]), 229
SUBSTRUC(par1,par2,par3), 230
SYSCMD(par), 231
SYSCMDNOECHO(par), 231
TABLE_INFO(par1, par2 [,par3]), 231
TEXT_DESTINATION, 232
TEXTONLY(par), 232
TEXTWIDTH(par), 232
TOC(par1 [,par2, par3, ..., par10]), 232
TYPE(name), 233
UERR(par1 [,par2]), 233
UNSTACK(name), 234
UPPERCASE(par), 234
VECLEN(name), 234
VTYPE(name), 234
X(par1 [,par2, par3, ..., par10]), 235
keyword, ALIGNMENT = kwd, 192
BOTMARGIN = length, 193
COLUMNS = expression, 189, 193
DEFCOLGAP = length, 191, 193, 194
DEFCOLWIDTH = length, 191, 193, 194
GAP = length, 192, 201
LEFTGAP = length, 193, 194, 207
LEFTMARGIN = length, 193
PAGESIZE = length structure, 193
RIGHTGAP = length, 193, 194, 228
RIGHTMARGIN = length, 193
TOPMARGIN = length, 193
WIDTH = length, 192, 234
layout macro, CHAP, 188
CONCL, 189
EEX, 195
EQT, 197
EX, 197
HEAD, 201
INTRO, 205
PART, 217
QT, 226
SIZE, 229
SUB, 229
TITLE, 232
macro, EPT, 196
OLDPT, 215
PT, 219
PTRESTORE, 224
PTSAVE, 224
TOCMACRO, 232
XMACRO, 235
system variable, ATTNS, 185
CURBOTMARGIN, 191
CURCOL, 191
CURCS, 191
CURFONT, 191
CURKEYBOARD, 191
CURLEFTMARGIN, 191
CURLINESPACE , 191
CURLP, 191
CURPAGEHEIGHT, 191
CURPAGESIZE, 191
CURPAGEWIDTH, 191
CURRIGHTMARGIN, 191
CURTABLINE, 191
CURTOPMARGIN, 191
CURUNDERLINE, 191
CURWORD, 191
FUNCTIONRC, 200
GARBAGECOUNT, 201
GARBAGESPACE, 201
INTPAGELNR, 205
INTPAGENR, 205
LAYNAME, 206
LINDENT, 207

LINDENTINDEX, 207
MACFLAG, 213
ODLOADED, 215
PDLOADED, 218
PGNTS, 218
RC, 226
RINDENT, 228
RINDENTINDEX, 228
SOURCELNR, 229
SYMSIZE, 231
THISPAGE, 232
TSIZE, 233
variable, ALIGNMENT = kwd, 184
BLANKLINE = length, 187
BOTMARGIN = length, 187
CAP = logical value, 187
CTR1 = 1, 190
CTR10 = 0, 190
CTR2 = 1, 190
CTR3 = 1, 190
CTR4 = 1, 190
CTR5 = 1, 190
CTR6 = 0, 190
CTR7 = 0, 190
CTR8 = 0, 190
CTR9 = 0, 190
DEFUNITS = kwd, 194
FOOTCONTINUE = expression, 200
FOOTCTR = expression, 200
FOOTDIVIDE = expression, 200
FOOTINDEX = expression, 200
FOOTSEP = expression, 200
FUZZ = expression, 200
GALLEY = logical value, 200
INPUTMODE = kwd, 204
LC, 207
LEFTINDENTS = length structure, 207
LEFTMARGIN = length, 207
LINESPACE = length, 207
MAXWORDSPACE = length, 214
MINHYPH = expression, 214
ODCHARACTERSET = expression , 215
ODFONT, 215
ODNAME = expression, 215
ODTSIZE, 215
PAGESIZE = length structure, 217
PARAIND = length, 217
PARASEP = length, 217
PDCHARACTERSET = expression, 218
PDNAME = expression, 218
PGNTLIMIT = expression , 218
PNCTR = expression, 218
PNHEAD = expression, 218
PNTRAIL = expression, 218
PRINTON = kwd, 218
PROOF = logical value, 219
PTCOUNTER = structure, 221
PTITEMGAP = length structure, 222
PTLEV = expression, 222
PTPOSTGAP = length structure, 223
PTPOSTINDENT = structure, 223
PTPOSTLEVELSTRING = string structure, 223
PTPOSTSTRING = string structure, 223
PTPREGAP = length structure, 223
PTPREINDENT = structure, 223
PTPRELEVELSTRING = string structure, 224
PTPRESTRING = string structure, 224
PTSEP, 225
PTTEXTINDENT = structure, 225
PTTEXTINDENTHANG = structure, 225
RIGHTINDENTS = length structure, 228
RIGHTMARGIN = length, 228
RUNPAR, 228
R0 = expression, 228
R1 = expression, 228
SENTSEP = length, 229
SERERRS = logical value, 229
SPLITSTRING = expression, 229
SYMMAX = expression, 231
TIMERLIMIT = expression, 232
TOCHEADER = expression, 232
TOCHEADERLIST = structure, 232
TOCINDEX = expression, 232
TOCMACROLIST = structure, 232
TOPMARGIN = length, 233
UC, 233
UNDERLINEDISPLACEMENT = length, 233
UNDERLINESTRING = expression, 233
UNDERLINEWORDSPACE = logical value, 233
UPPERCASEINPUT = logical value, 234
VERTALIGNMENT = kwd, 234
WORDSPACE = length, 234
XCOUNT = expression, 235

XHEADER = expression, 235
XHEADERLIST = structure, 235
XINDEX = expression, 235
XMACROLIST = structure, 235
XPARS = expression, 235
XPGPRINT = expression, 235
XPRINT = expression, 235

ERROR MESSAGES

'name' is not yet define. . ., 147
&FL is not yet defined. . . ., 110
&name is not defined. De. . ., 70
%Too many parameters to . . ., 171
'name' is . . ., 34
A NEWPAGE or PAGEEND com. . ., 50
A table entry with this . . ., 89
All RESERVEs must have a. . ., 227
Assignment to this item . . ., 123
Attempt to change a cons. . ., 123
Attempt to TAB past last. . ., 85
Bad parameter list forma. . ., 115
Both arguments of multip. . ., 100, 112
CASE index out of range . . ., 158
Character is not availab. . ., 30, 40
Contents of &FL included. . ., 110
Defined columns and thei. . ., 76
DELAY was pending but wa. . ., 73
Diagonal lines are not s. . ., 145
Emphasis is not availabl. . ., 38
End of file encountered . . ., 57, 156
ENDFOOTNOTE command does. . ., 57
Endless pointer chain de. . ., 108
FLOAT or FOOTNOTE is alr. . ., 56
FLOAT, FOOTNOTE, or KEEP. . ., 51
Index entry too long to . . ., 142
Index number *nn* contai. . ., 137
Input line longer than 2. . ., 97
Item assigned is STATIC.. . ., 126
Justification space requ. . ., 12
Line width exceeded in A. . ., 94
LNE is not defined. Igno. . ., 4
LOCATION ID is already d. . ., 144
Logical backspacing used. . ., 41
Missing continuation lin. . ., 27
Missing ENDIF for IF com. . ., 156
name is already defi. . ., 34
Negative horizontal spac. . ., 19
No reserve by that name.. . ., 66, 230
Non-line-breaking word s. . ., 30
Not allowed in this type. . ., 78
Number too large to sca. . ., 126
Number too large to scal. . ., 127
Number truncated to 4 de. . ., 99
Only one argument of add. . ., 100
Output Device is already. . ., 170, 216
PAGE, COLUMN, PAGEEND a. . ., 85
Requested horizontal spa. . ., 19
RESERVEs on page leave n. . ., 57, 169, 227
Result of expression can. . ., 127
String delimiter missed, 115
String delimiter missing. . ., 97
Table entry is longer th. . ., 169
Table entry is too long . . ., 85
Table of Contents number. . ., 133
The RESERVE, or FLOAT is. . ., 198, 227
The size of this page ma. . ., 198, 227
The TAB command is only . . ., 85
Too many levels specifie. . ., 142
TYPESIZE command not all. . ., 43
Unable to keep non-line-. . ., 11, 84, 85, 94
Unit SPUNCH was referenc. . ., 167, 168
Unmatched parentheses, 115
Value higher than maximu. . ., 125
Value lower than minimum. . ., 125
Value not found in value. . ., 125
Vertical size specified . . ., 198, 227
xxxx more than the width. . ., 84, 92
You cannot back up any f. . ., 66