A TEXT FORMATTER

by

Arndt von Staa

Research Report CS-74-21

Department of Computer Science

University of Waterloo

November 1974
A TEXT FORMATTER

by Arndt von Staa
University of Waterloo
October 1974
1. **Introduction.**

This text formatter is a second generation text formatter. It improves on the first generation formatter[3] by inclusion of some new capabilities, removal of some deficiencies found during the use of the first generation formatter and, finally, by reformulation of some of the algorithms. The main features of this new text formatter are:

i- to allow collecting figures and footnotes;

ii- to allow overprinting, underscoring and accentuation mark insertion by means of easy to use commands;

iii- to maximize the contents of a line by moving words from the input 'stream' to the lines. Words may be broken on user request if the line length is exceeded. The right margin is usually aligned;

iv- to allow both word shifting as in (iii) above, or to generate lines which are pseudo input record images;

v- to be completely parametric, i.e. any formatting parameter may be defined or redefined at execution time.

The text formatter has been written in SPITBOL. The input text to the formatter is a file created through some file editing system, e.g. WITS or WYLBUR, or by means of some utility.

This manual is intended as a reference manual and not as a tutorial guide. The author regrets the lack of mnemonicity of the text commands. Still, the author believes it to be a worthwhile investment to learn to use this text formatter.
2. Basic Concepts.

In this section we will define the terminology used throughout this text.

The input text consists of words and text commands. A text command is one of several well defined strings preceded by the escape character %. The text commands will be explained in detail in later sections. Words are strings of characters usually delimited by the blank character, or, in all cases, by end of input record or by delimiting text commands [%GH, %I, %P, %T, %U, %Z and %/].

The 8 last characters of an input record will always be stripped off. This is done in order to avoid problems when using text editors which introduce line numbers in the records.

The formatter may operate in either of the following two modes: formatted and unformatted. In the formatted mode, words are delimited by the end of card, delimiting text commands and one or more blanks. Words are also shifted to fill lines as much as possible. The input number of blanks between words is not taken into account. Usually right alignment will be performed. Right alignment is obtained by separating words by more than one blank. This extra blank introduction is such that blank 'streaks' in a page are avoided.

In the unformatted mode, words are delimited only by end of card and word delimiting text commands. Furthermore only one word per line is printed. In unformatted mode blank characters are considered in the same way as any other character. Unformatted mode is used when an output
format is desired which cannot be achieved in the formatted mode, for instance when building tables.

At any instant there is an active indentation descriptor. This indentation descriptor contains several parameters, which are used for the purpose of line alignment, spacing and output control.

There are three types of lines:

i- first line — this is the first line to be output when a new indentation descriptor becomes active [\$Inam text command, see section 3.6].

ii- paragraph line — this is the line to be printed after a paragraph text command [\$P see section 3.6]. If an indentation request is followed immediately by a paragraph command, the output line in between is null. Null lines are never output and never counted.

iii- continuation line — this is any line which is neither of the above.

A text block is a set of lines starting with a text block delimiting line, and ending with the line immediately preceding the next text block delimiting line. A text block delimiting line is any of first line or paragraph line, or a line output after a text block delimiting text command [\$TN, \$UN and \$ZN].

Lines of type first line may contain two types of words: primal words and normal words. Primal words are the first n≥0 words in the line. The value of n is defined in the indentation descriptor [see section 3.6]. Primal words are always concatenated one to the other leaving exactly one intervening blank. The string obtained in this way is
then right aligned at the *primal right margin*. There is no left alignment for primal words. A use of primal words \([n=1]\) can be seen in above indentations, where the roman numerals are instances of primal words. The remaining words in the first line and all words in paragraph and continuation lines are normal words. The first normal word in any of the three kinds of lines, will always be left aligned, regardless of the formatting mode.

There is only one *right margin* value per indentation descriptor. This value controls the size of all the three line types. The value corresponds to the rightmost print column of each line.

Each indentation descriptor must also provide the value for the *minimal end group*. This minimal end group is the set of physical lines at the end of a text block which must remain together on the same page. This parameter is used in order to avoid widow lines on the top of a page.

For lines of type first line and paragraph line, there is a *minimal page size* parameter defined. This parameter tells the minimum number of physical lines which must still be available on the current page. If this number of physical lines is not available, the current page is terminated and a new one is started. This parameter is used in order to avoid widow lines on the bottom of a page. This parameter can also be used to force a complete text block to be on the same page \(['A', see section 3.6].\)

The minimal end group MEG and the minimal page size MFS should satisfy following relation:

\[ MEG \leq MFS \]
The difference MPS-MEG defines the number of lines of the text block start, which are certainly going to be maintained on the same page. The effective minimal page size is this difference rather than the MPS value, since lines at the end of a text block may be stolen when pages are cut. Titles which are linked to the text, should be given MPS-MEG lines away from the first line to which they refer, or, when possible, after a line terminating command, e.g. $\&Z$.

For each of the line types, following parameters are defined:

i- **left margin** – this is the print column minus one, where the first character of the first normal word in the line, is to be placed. In the case of first lines, this parameter should be greater than or equal to the primal right margin parameter, otherwise the end portion of the primal word string is lost.

ii- **minimal left string** – this is the length of the leftmost portion of the line within which all words are separated by single spaces. If this parameter is larger than right margin minus left margin, no right alignment will be performed. This parameter is useful also when a line may point out at the left, in which case insertion of extra blanks would make the output appear less neat.

iii- **vertical space** – this parameter tells how many blank lines are to be inserted preceding a line of this type.

On each page there will be one or more *header lines* printed [the standard is 4]. The first line contains a
A Text Formatter

3. Text commands.

3.1 Introduction.

In this section we will describe all available text commands. First however, we will provide some general information.

There is no error checking performed on text commands. If any strange thing should happen to your output, check whether all text commands are valid and if they occur in a meaningful sequence. For instance, footnotes within footnotes might produce some results, but this seems not to be a meaningful sequence.

+ A physical page contains 66 lines at 6 lines per inch and 88 lines at 8 lines per inch.
The case of the text commands is important, that means that the text commands \$T and \$t are different text commands.

If a SPITBOL detected execution error should occur, an error message will be printed. This error message contains the rest of the input record and its sequence number, i.e., the last 8 characters of the input record.

The format is:

**ERROR**n1-STn2 | input record remainder string

where n1 is the SPITBOL error code and n2 is the statement number where the error occurred.

Some text commands initiate a special collecting mode, e.g., string collection, unformatted mode collection, figure and footnote collection. These commands normally require a special collection mode termination text command. This command is \$T. The portion starting at the special collection mode issuing command up to and including the \$T command, will be considered as one single command, regardless of whether or not there are text commands within the text collected in the special mode.

Some text commands delimit words [\$GH, \$I, \$P, \$T, \$U, \$Z and \$/]. Other text commands will insert a string into the word at the position of command occurrence [\$hex, \$JD, \$JS, \$N...\$T, \$urgeon, \$l]. All other commands do not affect the input and behave as if they where null strings.

Following is the meaning of the notation used by the text command descriptions:

(X)  X may occur optionally;
\{X|Y\} X or Y, not both, may occur optionally;

char a single character occurs. The set of which char is
to be a member is explained in the text command
description;

string a sequence of characters may occur. Usually a special
collection mode termination command will be required
[\&T]. When string is being collected, only
underscore, constant text overprint, wordbreak
[always acts as a null text command], \$hex, \$z, input
record concatenation \$V, \$JD, \$JP and \$JS are valid
text commands. In some contexts the overprinting and
underscoring will be lost. The string is always
collected in the unformatted mode, i.e. blanks are
normal characters.

text any portion of normal text formatter input text,
always followed by a \&T command;

X X is a variable and its valid values are described in
the text command description;

X X is a character constant and must be literally
present. The character blank is denoted by \&.

3.2 Miscellaneous commands.

\$z this text command inserts the character \$ into the
word at the place where it occurs. For example:

abc\$def is printed: abc\$def.

\$hex this text command inserts the character denoted by the
two hexadecimal digits \([\text{hex}=\text{two characters of the set }\{0123456789\text{ABCDEF}\}]\) into the word at the place where the command occurs. For example:

\[
\alpha \text{AAD} \# 3B \sigma 3C \quad \text{is printed:} \quad a \alpha \beta \gamma
\]

Figure 3.1 shows a table of the special characters available on the UN print train. Only characters which can not directly be input through the IBM 2741 terminal are listed.

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
<th>ASCII</th>
<th>#</th>
<th>$</th>
<th>£</th>
<th>€</th>
<th>£</th>
<th>£</th>
<th>£</th>
<th>£</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>`80</td>
<td>0</td>
<td>5F</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>`31</td>
<td>1</td>
<td>31</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>`32</td>
<td>2</td>
<td>32</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>3</td>
<td>`33</td>
<td>3</td>
<td>33</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>`34</td>
<td>4</td>
<td>34</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>`35</td>
<td>5</td>
<td>35</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>6</td>
<td>`36</td>
<td>6</td>
<td>36</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>7</td>
<td>`37</td>
<td>7</td>
<td>37</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>8</td>
<td>`38</td>
<td>8</td>
<td>38</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>9</td>
<td>`39</td>
<td>9</td>
<td>39</td>
<td>22</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

Figure 3.1 Table of hexadecimal representations of the special characters in the UN train.

\[\textbf{GRnn}\]

this command sets the left margin parameter of the record number. The record number is defined as being the last eight characters of the record. In case the text of the line is longer than this left margin, no
record number will be output for this line. Originally this parameter is set to 90.

\$V

This command terminates the current input record, however, it does not terminate the current word. Thus it might be thought of as if it were an input record concatenation command. If the current mode is formatted, the first character of the new input record should not be blank, i.e. \$. [see \$P command, section 3.6]. If a blank is required at that point, it should be provided by hexadecimal inclusion, \$40. This command is very useful when collecting text in unformatted mode and the pseudo record image description does not fit completely on one input record. Observe though, that this command allows the generation of very long words. If it happens to generate a word of more than 130 characters, this word will be split into several consecutive single spaced lines. To counter this problem when in unformatted mode, place a \$/ [see below] command immediately after the last character of the word obtained by \$V concatenations. For example, the input text lines:

abc def ghi klm\$Vabcdefghijk
nop qrs tuv

are printed:

abc def ghi klmnop qrs tuv

and the string 'abcdefghijk' is lost.

\$/

this command terminates both the current word and also the current input record. The same input example as above, replacing \$V by \$/, yields:
A Text Formatter

abc def ghi klm
nop qrs tuv

%T{N|Y}

this text command terminates special collection modes. All commands which start a special collection mode, must be followed by this command [%JS, %JP, %N, %S and %U, the commands %K... will automatically append a %T to the first character ) found]. Recursive start of special collection mode is allowed. The user must ensure, however, that this sequence is meaningful. For example, unformatted mode within a figure is a valid sequence.

The form %TN has meaning only if unformatted mode is being terminated. In this case the command flags also the end of text block [see section 2].

%char %{

where char is any of the names of a word break descriptor. This command is used to flag a valid word break position. If the word overflows the current line, it will be broken at the point indicated, provided that the break string and the initial portion of the word fit in the line. There may be more than one break indication per word. In this case the rightmost break position which satisfies the break condition will be used. Underscores and overprints will be taken into account, i.e. the underscore and overprint strings will also be "hyphenated" and continued on the next line. Observe that word break commands occurring in words which do not overflow a line are null commands.
$\texttt{KBchar('string')}$

define or redefine a new word break descriptor.

\texttt{char} is the word break descriptor name. It must be a single character, and it must be disjoint from any of the text command first characters, the underscore, the word or line fill and the constant text overprint descriptor names.

\texttt{string} is the string which is to be appended to the first portion of the broken word. \texttt{string} may be the null string.

Figure 3.2 shows the system defined word break descriptors, and the commands used to define them.

\begin{tabular}{lll}
\texttt{char} & \texttt{string} & \texttt{defining command} \\
- & - & $\texttt{KB-('-')}$ \\
, & & $\texttt{KB,('')}$ \\
\end{tabular}

Figure 3.2 Table of system defined word break descriptors.

The following lines of input:

$\texttt{EXIT}(0,0,40,0,0,30,0,0,0,30,0,0,30,0,0,30,0)$ $\texttt{EXIT}$

$\texttt{EXE}(1,'%')$

ab\%-cd eff,gh ij kl\%-mn\%-opqr st\%,uvwx

$\texttt{bxy\%zabcdefg}$

produce following output:

\begin{verbatim}
abcd efg h
ij klm
opqr st
uvwx %
zabcdefg
\end{verbatim}

$\texttt{JDI}$

this text command inserts the date into the word at the place of occurrence. The date is the SPITBOL date, i.e. the string mm/dd/yy where mm is the number of the
month, dd is the day and yy are the last 2 digits in the year. For example:

...$JD... is printed: ...10/15/74...

$JS\texttt{string}$T

in this command \texttt{string} must be a valid SNOBOL4 object, i.e. right hand side. The expression \texttt{string} should deliver a value of type \texttt{string} or convertible to \texttt{string}. The result of the evaluation of \texttt{string} will be inserted into the word at the place of the command occurrence. For example:

\begin{verbatim}
abc$JS DUPL('**',5) $Tefg is printed:
abc*****efg
\end{verbatim}

$JP\texttt{string}$T

in this command \texttt{string} must be a valid SNOBOL4 program section. When ending, this program should either flow through the end, or use the label \texttt{RETURN}, in order to return control to the text formatter. It is also advised that this command be used only after full understanding of the text formatter program has been achieved. After \texttt{string} has been collected, the program represented by \texttt{string} is compiled and executed [see \texttt{CODE} function in SNOBOL4]. For example the string

\begin{verbatim}
abc$JP ESCAPE FN = 'hm hm hm hm' $Tdef
\end{verbatim}

generates the next strange output

\begin{verbatim}
abchm hm hm hmdef
\end{verbatim}

$J$

insert the current page number into the output text at the place of occurrence. The current page number is not necessarily the page number output at the top of
A Text Formatter

the physical page. This is a consequence of the existence of the page number redefining command \#G# [see section 3.7]. Furthermore if the command \#J# is given within a minimal end group, it may happen that the line containing the reference is printed on the next page.

Example:
abcd\#J#efg is printed: abcd14efg

\#YIstring\%
assign the current page number to string. Recall that the current page number is not necessarily the number printed on the page containing the text surrounding \#YI, c.f. \#J# above. The numbers will agree if the relation \text{MEG}<\text{MPS} [see section 2.] holds, and if the command \#YI is given so that it follows the complete text of a text block starting line.

\text{string} is delimited by a blank, i.e. \text{ }. This blank is automatically deleted from the input text. Care should be taken, so that \text{string} is not equal to any of the variables used by the text formatter program itself. This will be the case if \text{string} does not follow the \text{SNBCBL} variable name conventions, or if the string \text{string} consists of lowercase characters.

\#YRstring\%
insert the value associated with \text{string} into the text. This text command retrieves the value associated with \text{string} by a \#YI text command.

Example:
abc\#YIxx def\#YRxx ghi is printed: abcdef14ghi
A Text Formatter

The text commands \#YI and \#YR have been designed to facilitate the creation of tables of contents, or indices. A practical example of their use is appendix 2. Observe though, that such tables of contents can only be provided after all the text referred to by the table of contents has already been read in.

\%char, \#wchar or \%lchar  \$\{[w|l]{i}\}

where char is any of the word or line fill descriptor names.

Used in the form \%char, the current word will be appended by a string which length is the length defined by the descriptor, and which characters are taken from those defined in the descriptor.

Used in the form \#wchar, the current word will be filled with the string defined in the descriptor up to the length defined by the descriptor. Used in the form \%lchar, the current line will be filled, under the assumption that words are single spaced. If the filling string is blank, \%lchar corresponds to a tab to the position defined by the length parameter in the descriptor char.

Observe that the filling of the line is defined relative to the physical left margin. Thus the line fill size corresponds to some virtual right margin. The word fill is defined relative to the start of the word. In unformatted mode the word fill is then relative to the line left margin.

The filling string may be of any length. If the portion to be filled does not hold an integer multiple of the filling string, the first portion will be
filled with a sufficiently long trailing substring of the filling string.

\texttt{\$KFchar('string',size)}

define or redefine a word or line fill descriptor.

\texttt{char} is the word or line fill descriptor name. It must be a single character, and must be disjoint from the text command first characters, the word break, the underscore and the fixed text overprint descriptor names.

\texttt{string} is any nonnull string which will be used to fill the line or word up to the specified length.

\texttt{size} is the length up to which the line or word will be filled.

Figure 3.3 shows the system defined word or line fill descriptor, and the command used to define it.

\begin{center}
\begin{tabular}{ccc}
\texttt{char} & \texttt{string} & \texttt{size} & \texttt{defining command} \\
\texttt{\$KFf(',',69)} & \texttt{1} & \texttt{.} & \texttt{69} \\
\end{tabular}
\end{center}

\begin{center}
Figure 3.3 Table of the system defined word or line fill descriptor.
\end{center}

For some documents, the characters available on the print chain being used are insufficient. Spacing will then be required wherever such a character is to be used. The character will, then, be introduced later by means of a typewriter for instance. For example the declaration:

\texttt{\$KFz('\$70',1)}
defines a new line and word fill descriptor. This descriptor will reserve one single space, and print the character \texttt{'} in this space. It is then quite easy to spot
A Text Formatter

and introduce the required character by means of a typewriter.

Examples:
£Z1 ab cd e£lig ij is printed:
ab     cd
ef...£lig ij
e£wig ij is printed:
ab     cd
e £wig ij
Definition of 'z'
£Fz('12345',15)
£Z1 a£z ab£z abc£z is printed:
a123451234512345 ab123451234512345 abc123451234512345
£Z1 a£lz ab£lz abc£lz is printed:
a12345 ab abc
£Z1 a£wz ab£wz abc£wz is printed:
a23451234512345 ab3451234512345 abc451234512345

3.3 Underscoring and Overprinting.

£char or £&char £{|@}|{b|d|n|.*}
where char is one of the underscore descriptor names, is the command which controls underscoring. If char is the character '. ' [point] no further underscoring will be performed. If any underscoring was in progress when a new underscore command is issued, the underscore in progress will be terminated at the point where the new one is issued. Thus the command £. has the same effect as an underscore stop command.

If the command is of the form £char, underscoring will automatically be terminated at the end of the
word. Care must be taken to not succeed a command by an empty word, i.e. blank, for otherwise no underscoring will occur. Observe that in unformatted mode the whole line is a single word. Thus, explicit underscore termination must be provided.

If the command is of the form `&char`, underscoring can only be terminated by the end of underscore command `&`, i.e. the underscoring remains active over word boundaries.

Underscoring is done only under the affected word, thus blanks between words will not be underscored. Observe that, in unformatted mode, blanks are part of the word and, consequently, they will be underscored.

`&uchar('string', 'assoc')`
define or redefine a new underscore descriptor.

`char` is the underscore descriptor name. It must be a single character, and must be disjoint from the text command first characters, the word break, the word or line fill and the constant text overprint descriptor names.

`string` this is the string to be used to underscore the word. It should be exactly one character long; otherwise the underscoring might overflow the word boundaries.

`assoc` is the line association of the underscore character. If `assoc=N`, the underscore line will be associated with the next physical print line. This is needed if the character used to underscore is not a true underscore character, such as `'*'`. If
assoc=C, the line association of the underscore is the current line.

Figure 3.4 shows the system defined underscore descriptors and the commands used to define them.

<table>
<thead>
<tr>
<th>char</th>
<th>string</th>
<th>assoc</th>
<th>defining command</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>_</td>
<td>C</td>
<td>$%Un('_','-','C')</td>
</tr>
<tr>
<td>d</td>
<td>_</td>
<td>C</td>
<td>$%Ud('_'EB','C')</td>
</tr>
<tr>
<td>b</td>
<td>~</td>
<td>N</td>
<td>$%Ub('_'EC','N')</td>
</tr>
</tbody>
</table>

Figure 3.4 Table of the system defined underscore descriptors.

Some examples:

ab _word xyz is printed: ab word xyz
ab _nde fg_h is printed: ab de fg hi
$\%Un('*','N')
isn't th's*at nice%? is printed: isn't that nice?

$\%Un

\%[a|c|f|g|s|t|u]

where char is one of the constant text overprint descriptor names, causes a constant text to be overprint on the current word starting at the character preceding the command. This command is used mainly to introduce accentuation marks into the text.

$\%Kchar

define or redefine a new constant text overprint descriptor.

char is the constant text overprint descriptor name. It must be a single character and it must be disjoint from the command first characters, the word break,
the word or line fill and the underscore descriptor names.

*string* is the string used to overprint. *string* might be of any length, however, care should be taken that it never overflows word boundaries.

*assoc* has the same meaning as *assoc* in the underscore descriptor.

Figure 3.5 shows the system defined constant overprint descriptors and the text command used to define them.

<table>
<thead>
<tr>
<th>char</th>
<th>string</th>
<th>assoc</th>
<th>defining command</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>'</td>
<td>C</td>
<td>$KOa('OF', 'C')</td>
</tr>
<tr>
<td>g</td>
<td>'</td>
<td>C</td>
<td>$KOg('2B', 'C')</td>
</tr>
<tr>
<td>f</td>
<td>^</td>
<td>C</td>
<td>$KOf('08', 'C')</td>
</tr>
<tr>
<td>t</td>
<td>~</td>
<td>C</td>
<td>$KOt('29', 'C')</td>
</tr>
<tr>
<td>u</td>
<td>'</td>
<td>C</td>
<td>$KOu('11', 'C')</td>
</tr>
<tr>
<td>c</td>
<td>.</td>
<td>C</td>
<td>$KOC('09', 'C')</td>
</tr>
<tr>
<td>s</td>
<td>/</td>
<td>C</td>
<td>$KOs('/', 'C')</td>
</tr>
</tbody>
</table>

Figure 3.5 Table of the system defined constant overprint descriptors.

Examples:

fac%cade is printed: façade

$KC('|', 'N')

up%point% is printed: uppoint

string1#base-string@[#] over1[#] over2 ... overn#string2

this command generates a variable overprint string.

The resulting word is obtained from the following concatenation:
any or both of string1 or string2 may be empty. If the character # following the @ overprint delimiter character is present, the line association of the overprint string is to the next line. The overprint strings over1, over2, ..., overn, are all placed at the first character position in the base-string. The base-string and the over1 strings are all subject to the same restrictions as for string described in section 3.1. If no overprint string is present, i.e. @ does not appear within the #...# text, the #string# text command can be viewed as an unformatted mode string collecting command.

For example:

```
abc##def@/a#/||gghi is printed:   abc####gghi
||
The string:   #user defined separation   ###
is printed:   user defined separation   ###
```

### Figures and footnotes.

Figure and footnote commands do not delimit words. The footnote command automatically inserts the footnote reference mark into the word. Figure and footnote text should be given in the place where they are supposed to occur in the output text. Thus, if some word makes a reference to a footnote, this footnote should be provided at the place in the word where the footnote reference should occur. For example:

foot#N This is a footnote. #N This generates following word and the footnote reference below:
__________________________
+ || is the concatenation operator.
footnote

Figures are placed in the page where they fit. If the text for a figure is too large for the remaining part of the page, it will be placed in a waiting queue in order to be printed on one of the succeeding pages. The order with which figures occur in the text is strictly obeyed in the output of the text. If a figure is too large to fit on a complete page, it will be broken into several pages; the first page will be occupied completely by the figure. However, no text block formatting occurs when a figure is broken, thus the user is responsible for end of page formatting.

Footnotes which are too large are also broken. A footnote must have 3 lines at least in order to be broken. Furthermore, there must be at least as many lines in the remainder of the page as needed to contain the line referring to the footnote and two lines of the footnote. One limitation exists, however, no line may refer to more than one footnote. Footnotes may be broken into several portions such that there is footnote continuation text over more than one page. This feature and the figure break feature allow for very long footnotes and figures, but they also allow missing $T$ commands to pass unnoticed. Thus if there is any problem with memory overflow during execution, check if all of the special collection mode start commands have their necessary termination command $T$.

A footnote goes to a text page only if the line referring to it is on the same text page. Thus it is possible to have figures referring to footnotes, without that the synchronism is violated.
\%S

this command starts the collection of a figure text. Within a figure text any text command may be issued, even definition of a new figure. The status of the formatter at the point of the command \%S is preserved. The indentation descriptor which is normally active when collecting figure text, is the \%FIG descriptor [see section 3.6]. It may be redefined by the user at any time. The automatic spacing increase is normally 0, thus figures are single spaced text.

Figures are null string commands, i.e. they do not affect the word in progression when the figure collection command is found. After the corresponding \%T command has been found, the status of the text formatter is reset to what it was when the collection of the figure text began.

\%N

this command starts the collection of a footnote text. Within the footnote text, any text command may be issued. The status of the formatter at the point of the \%N command will be preserved. The footnote reference character will be concatenated to the word being built. This character will also be inserted into the input text such as to form the first word to be collected by the footnote.

The indentation descriptor which is normally active when collecting footnote text is the \%IFOOT descriptor [see section 3.6]. It may be redefined by the user at any time. The automatic spacing increase is normally 0, thus footnotes are collected as single spaced text.
Several parameters govern the collection of figures and footnotes. Two have already been mentioned, these are the figure indentation descriptor \$IFIG and the footnote indentation descriptor \$IFOOT.

\$HFstring\$T
this command redefines the figure delimiter line which is always used to surround the figure text. Its normal value can be seen in the figures within this text. When several figures are placed on a page, one following the other, only one figure delimiter line is output. Thus if several figures in a close sequence are given, they should be given one after the other without intervening text.

\$G1nn
sets the figure start spacing parameter to \$nn. All figures are automatically spaced from the preceding text line by a number of lines equal to this parameter [normally 0].

\$G2nn
sets the spacing parameter between the last line in the figure text and the figure delimiting line to \$nn [normally 0].

\$G3nn
sets the spacing parameter between the figure delimiting line and the next normal text line to \$nn. The spacing will occur only if the value of this parameter is larger than the the normal spacing of the current line being built [normally 1].
this command sets the footnote delimiting line to string. Its normal value can be seen in the footnote reference containing pages in this text.

this command sets the footnote marker list to string. Whenever a new footnote reference is issued on a page, a footnote counter per page is increased by one. This counter is then used to access a character out of the footnote reference string. When a new page is started, this counter is set to zero. This might cause a same marker to be used twice on a page, if the line containing the footnote is passed on to the next page due to lack of space. This is an algorithm bug, but it will be noticeable only if the footnote density is high. It can be countered by using conditional page ending command %GZ. The normal value of this list is $12345$.

this command sets the maximum number of footnote lines per page. If a footnote has to be broken due to being too large, there will always be less than nn lines of footnote text on the next pages, regardless of the size of the footnote text.
3.5 Unformatted mode.

\$U\{N\} pseudo record images \$T\{N\}

this command starts collection in the unformatted mode. The only basic difference between formatted mode and unformatted mode is that in unformatted mode the character blank is used as a normal character. Thus the user is able to specify his own spacing requirements. Furthermore, in unformatted mode only one word per line is output. Notice however, that this word might be quite large due to the inclusion of blanks.

The standard line parameters are those of the continuation line in the current active indentation descriptor. Within unformatted mode any command may be issued. This ability of using any text command makes counting of characters quite difficult if null string commands, or commands inserting strings, are issued.

When a \$U command is found, the current line is terminated, the remaining of the input record is erased and the input starts from a new record. If also \$N is specified, i.e. \$UN, the next line will be a text block delimiting line. The parameters used to verify if the new text block can be placed on the page are those from the paragraph line in the currently active indentation descriptor.

An example of the use of unformatted mode can be seen in the figure texts within this text.
3.6 Text line affecting commands.

\%P or first input record character % in formatted mode
this command terminates the current line, and sets the
next line type to paragraph line. It acts thus as a
paragraphing command. It may be issued in unformatted
mode, but then only in its explicit form \%P rather
than first character blank. This command delimits text
blocks.

\%Inam
this command terminates the current line. It then sets
the indentation descriptor with name nam to active and
the next line to be printed to firstline. The name nam
is any string not containing blank or %. Care must be
taken to assure proper name nam delimitation. If a non
existing indentation request is issued, only line and
text block termination will occur. The next line will
be continuation line in this case, and the string nam
will remain in the text.

\%XInam(pw,ir,rm,me,vsf,lmf,ssf,tbf,vsp,lmp,ssp,tbp,vsc,lmc,ssc)
this command defines a new indentation descriptor with
name nam. The parameters are:
pw— the number of primal words
ir— the primal right margin
rm— the right margin
me— the minimal end group size
vs— the vertical spacing previous to a line of this
type. First vsf; paragraph vsp; continuation vsc
lm- the left margin of a line of this type

ss- the minimal single spaced left string of a line of this type

tb- the minimal page remainder for a line of this type. If some integer value, it is the minimal number of lines which must remain free on the current page, in order that a line of this type be placed on the current page. If this value is the character 'A', the text block must go to the page as a single unit.

Figure 3.6 shows the system defined indentation descriptors and the defining commands used to define them.

Hint for letters. Start text with:

\%EIN(0,0,74,3,1,38,100,5,1,8,100,5,0,3,100) \%IN \%GS0

This will cause a non right aligned single spaced text to be printed. Lines of type first line may be used for date and signature lines.

\%Z{L|N} {I} {R} \%n

This command terminates the current line and, then, starts a new line of type continuation line which will be preceded by \%n blank lines. If the value of \%n is null, i.e. has not been provided, the vertical spacing of the line following this text command will be the continuation line spacing added to the current spacing parameter [see \%GS]. If the option L, i.e. \%ZL, is given, the line will be right justified, otherwise no justification occurs. If the option N, i.e. \%ZN, is given, this command also terminates the current text block. The parameters used to initialize the next text block are those of the paragraph line of the currently
<table>
<thead>
<tr>
<th>parameter</th>
<th>indentation descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td><strong>pw</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>ir</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>rm</strong></td>
<td>68</td>
</tr>
<tr>
<td><strong>me</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>vsf</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>lmp</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>ssf</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>tbf</strong></td>
<td>18</td>
</tr>
<tr>
<td><strong>vmp</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>lmp</strong></td>
<td>14</td>
</tr>
<tr>
<td><strong>ssp</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>tbp</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>vsc</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>lmc</strong></td>
<td>9</td>
</tr>
<tr>
<td><strong>ssc</strong></td>
<td>0</td>
</tr>
</tbody>
</table>

\$KIN(0, 0, 68, 4, 4, 4, 0, 18, 1, 14, 0, 5, 0, 9, 0)

\$KIR(1, 13, 68, 4, 1, 15, 0, 5, 1, 20, 0, 5, 0, 15, 0)

\$KIL(1, 16, 68, 4, 1, 18, 0, 5, 1, 23, 0, 5, 0, 18, 0)

\$KID(1, 19, 68, 4, 1, 21, 0, 5, 1, 26, 0, 5, 0, 21, 0)

\$KIDF(2, 17, 68, 4, 1, 18, 0, 5, 1, 23, 0, 5, 0, 18, 0)

\$KIFOT(1, 7, 65, 0, 0, 8, 0, 0, 0, 13, 0, 0, 0, 8, 0)

\$KIFIG(0, 0, 68, 0, 0, 6, 0, 0, 0, 16, 0, 0, 0, 11, 0)

**Figure 3.6 Table of system defined indentation descriptors.**

active indentation descriptor. If the option I, i.e. \$ZI or \$ZNI, is used, \(n+1\) blank lines are output immediately. This is needed if there may be vertical
spacing redefining commands [%GH, %GZ, %I, %P, %T, %U, %Z, %/] following the %Z command. If the option R is given, i.e. %ZR, the number of lines to be skipped nn will be increased by the current spacing parameter.

At the top of pages, vertical spacing is always set to zero. If vertical spacing is needed from the top of a page, the sequence %Z10 %Z1n-2 will satisfy. Observe also that the remainder of spacings in a page will be set to zero if a new page is begun.

%GZnn[R]
This command sets the vertical spacing of the line currently being built to nn. This command does not terminate the line being built, nor does it set the line type parameter. If the option R is given, i.e. %GZnnR, the number of lines to be skipped will be increased by the current spacing parameter.

3.7 Page affecting commands.

%GHnn
set the number of page header lines to nn≥0 [usually 3]. It terminates also both the current line and the current page. The number nn does not count the first title line. If nn is not a valid integer, this command takes the degenerate form of a page ending command, however, no page will be generated if the current page is empty. In the degenerate case footnotes and partial figures are not lost.

%GCnn
if there are less than nn available lines in the page remainder, the current page will be terminated. The
line in progression is not placed on the current page previous to the termination. Also the line type parameter is not modified.

\%GDnn

set the page depth to nn [usually 58]. It only modifies the page depth if the value of nn is larger than the number of header lines [see \%GH command above]. This text has been printed at 8 lines per inch and with a page depth of 68.

\%GSnn

set the automatic line space increase to nn≥0 [usually 1]. This parameter is used to control single and double spacing. Whenever a new line is begun, this parameter will be added to the vertical spacing parameter of the line type begun. However, explicit spacing commands overwrite this spacing of the current line [e.g. \%Znn and \%GZnn].

\%G\#nn

set the page number to nn. If nn is empty, the page number will not appear on the page and will not be increased automatically for each new page. This command might be issued anywhere in the text and will set the number of the page currently being built.

\%HAstring\%T

This command will set the page number follower title to string. It may be issued anywhere in the text and will affect the page currently being built. There may be no overprinting or underscoring command in string.
\[ \text{\texttt{\textbf{string}}} \]

This command sets the \texttt{n}th header line to \texttt{string}, i.e., it sets titles and subtitles. The subtitles in the page header lines \( i \geq n \) are set to blank. The first line in the page header is \( n = 0 \). Overprints underscoring are all valid within \texttt{string} and will be output on every page.

4. **JCL needed to run the program**

Following is the JCL needed to run this text formatter:

```
//name JOB account...
// EXEC SPITBOL,REGION.GO=200K,PARM.GO='R=15K'
//GO.SYSIN DD DSN=file-name
// DD *
```

In \texttt{file-name} there must be the source text of the text formatter. It takes less than 0.02 of a minute to compile the program.

Normally a text page, double spaced, will take slightly less than 0.01 of a minute to be processed.

5. **References**


Appendix 1.

The input text for the first 3 pages of this manual are exactly as follows:

This text formatter is a second generation text formatter. It improves on the first generation formatter by inclusion of some new capabilities, removal of some deficiencies found during the use of the first generation formatter and, finally, by reformulation of some of the algorithms. The main features of this new text formatter are:

- Allow overprinting, underscoring and accentuation mark insertion by means of easy to use commands;
- To maximize the contents of a line by moving words from the input 'stream' to the lines. Words may be broken on user request if the line length is exceeded. The right margin is usually aligned;
- To allow both word shifting as in (iii) above, or to generate lines which are pseudo input record images;
- To be completely parametric, i.e. any formatting parameter may be defined or redefined at execution time.

The text formatter has been completely written in SPITBOL. The input text to the formatter is a file created through some file editing system, e.g. WITS or WYLBUR, or by means of some utility.

This manual is intended as a reference manual and not as a tutorial guide. The author feels very sorry for the lack of mnemonic of the text commands. Still, the author believes it to be a worthwhile investment to learn to use this text formatter.

In this section we will define the terminology used throughout this text.

The input text consists of words and text commands. A text command is one of several well defined strings preceded by the escape character. The text commands will be explained in detail in later sections. Words are strings of characters usually delimited by the blank character, or, in all cases, by end of input record or by delimiting text commands AD%GH, EI, EP, HA, HZ and FEB.

The 8 last characters of an input record will always be stripped off. This is done in order to avoid problems when using text editors which introduce line numbers in the records.

The formatter may operate in either of the following two modes:

- Unformatted or unformatted. In the unformatted mode, words are delimited by end of card, delimiting text commands and one or more blanks. Words are also shifted to fill lines as much as possible. The input number of blanks between words is not taken into account. Usually right alignment will be performed. Right alignment is obtained by separating words by more than one blank. This extra blank introduction is such that blank 'streaks' in a page are
Appendix 2.

Summary of commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%00 ... %FF</td>
<td>-8</td>
<td>hexadecimal character inclusion.</td>
</tr>
<tr>
<td>%GCn</td>
<td>-30</td>
<td>skip to a new page if less than n lines remain on the current page. Lines are not terminated. This command guarantees at least n lines on the page.</td>
</tr>
<tr>
<td>%GDn</td>
<td>-31</td>
<td>set page depth to n.</td>
</tr>
<tr>
<td>%GHn</td>
<td>-30</td>
<td>terminate the current line. Skip to a new page if the current page is not empty. If 0&lt;5n&lt;9, set the number of header lines to n.</td>
</tr>
<tr>
<td>%GNn</td>
<td>-25</td>
<td>set the maximal footnote size per page to n.</td>
</tr>
<tr>
<td>%GRn</td>
<td>-31</td>
<td>set the record number left margin to n.</td>
</tr>
<tr>
<td>%GSn</td>
<td>-30</td>
<td>set the line spacing to n. [0 is single spaced, 1 is double spaced].</td>
</tr>
<tr>
<td>%GZn</td>
<td>-30</td>
<td>set the spacing preceding the current line to n. Do not terminate the current line.</td>
</tr>
<tr>
<td>%G#n</td>
<td>-31</td>
<td>set the page number of the current page to n.</td>
</tr>
<tr>
<td>%GIn</td>
<td>-24</td>
<td>set the spacing preceding the figure delimiter line to n.</td>
</tr>
<tr>
<td>%G2n</td>
<td>-24</td>
<td>set the spacing after the figure text and preceding the figure delimiter line to n.</td>
</tr>
<tr>
<td>%G3n</td>
<td>-24</td>
<td>set the spacing after a figure delimiter and the text succeeding the figure to n.</td>
</tr>
<tr>
<td>%HAstring#T</td>
<td>-31</td>
<td>set the page number follower title to string.</td>
</tr>
<tr>
<td>%Hstring#T</td>
<td>-24</td>
<td>set the figure delimiter string to string.</td>
</tr>
<tr>
<td>%Hmarker#T</td>
<td>-25</td>
<td>set the footnote marker string to string.</td>
</tr>
<tr>
<td>%Hndelimiter#T</td>
<td>-25</td>
<td>set the footnote delimiter string to string.</td>
</tr>
<tr>
<td>%Hnstring#T</td>
<td>-32</td>
<td>set the title of the header line 0&lt;5n&lt;9 to string. Set all titles m&gt;t&gt;n to null.</td>
</tr>
<tr>
<td>%Inam</td>
<td>-27</td>
<td>terminate current line and text block. Turn nam into the current indentation descriptor.</td>
</tr>
<tr>
<td>%JD</td>
<td>-12</td>
<td>insert the SPITBOL date into the text.</td>
</tr>
<tr>
<td>%JSstring#T</td>
<td>-13</td>
<td>obtain and execute the SPITBOL program section string.</td>
</tr>
<tr>
<td>%JSstring#T</td>
<td>-13</td>
<td>obtain, evaluate and place into the text of the current word, the SPITBOL object string.</td>
</tr>
<tr>
<td>%K#</td>
<td>-13</td>
<td>insert the current page number into the text.</td>
</tr>
<tr>
<td>%Kbchar(....)</td>
<td>-12</td>
<td>definition of the word break descriptor char.</td>
</tr>
<tr>
<td>%Kfchar(....)</td>
<td>-16</td>
<td>definition of the word or line fill descriptor char.</td>
</tr>
<tr>
<td>%Knam(....)</td>
<td>-27</td>
<td>definition of the indentation descriptor nam.</td>
</tr>
<tr>
<td>%Kochar(....)</td>
<td>-19</td>
<td>definition of the constant text overprint descriptor char.</td>
</tr>
<tr>
<td>%Kuchar(....)</td>
<td>-18</td>
<td>definition of the underscore descriptor char.</td>
</tr>
<tr>
<td>%Ntext#T</td>
<td>-23</td>
<td>generate a footnote with text text.</td>
</tr>
<tr>
<td>%P</td>
<td>-27</td>
<td>terminate the current line and text block. Initiate a new paragraph.</td>
</tr>
<tr>
<td>%Snext#T</td>
<td>-23</td>
<td>generate a figure with text text.</td>
</tr>
<tr>
<td>%T</td>
<td>-11</td>
<td>special collection mode termination command.</td>
</tr>
<tr>
<td>%Utext#T</td>
<td>-26</td>
<td>collect text in unformatted mode.</td>
</tr>
<tr>
<td>%V</td>
<td>-10</td>
<td>terminate the current input record, proceed from the next input record without signalizing end of record.</td>
</tr>
<tr>
<td>%Ystring#</td>
<td>-14</td>
<td>associate the current page number with string.</td>
</tr>
<tr>
<td>%Yrstring#</td>
<td>-14</td>
<td>insert the value associated with string into the text.</td>
</tr>
</tbody>
</table>
%Zn -28- terminate the current line and set the spacing for the next line to n.
&/ -10- terminate the current input record and signal end of record.
&#string# -20- collect string in unformatted mode, and generate overprints for substrings initiated with @.
&g -8- insert a % into the text.
&{a|c|f|g|s|t|u} -19- constant text overprint.
&{&}{b|d|n|*} -17- underscore.
&{l|w}{i} -15- word or line fill.
&{,|-} -11- word break.
first column blank in unformatted mode, see %P.
## Appendix 3.

### Hexadecimal correspondence tables.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>l</td>
<td>i</td>
<td>i</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>t</td>
<td>0</td>
<td>3</td>
<td>d</td>
<td>f</td>
<td>e</td>
<td>G</td>
<td>@</td>
<td>`</td>
<td>(</td>
<td>+</td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>/</td>
<td>*</td>
<td>#</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>!</td>
<td>$</td>
<td>*</td>
<td>)</td>
</tr>
<tr>
<td>6</td>
<td>/</td>
<td>t</td>
<td>o</td>
<td>d</td>
<td>p</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>^</td>
<td>£</td>
<td>%</td>
<td>_</td>
<td>&gt;</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>$</td>
<td>}</td>
<td>\</td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>a</td>
<td>l</td>
<td>r</td>
<td>[</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>j</td>
<td>r</td>
<td>]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>+</td>
<td>)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>`,</td>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>\</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>-</td>
<td>'</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A Text Formatter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>B0</td>
<td>AD</td>
<td>BD</td>
<td>BE</td>
<td>AA</td>
<td>0F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>B1</td>
<td>8B</td>
<td>9B</td>
<td>AE</td>
<td>3B</td>
<td>08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>B2</td>
<td>AC</td>
<td>BC</td>
<td>8C</td>
<td>3C</td>
<td>2B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>B3</td>
<td>AB</td>
<td>BB</td>
<td>9E</td>
<td>41</td>
<td>0E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>B4</td>
<td>FA</td>
<td>FB</td>
<td>62</td>
<td>42</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>B5</td>
<td>8F</td>
<td>63</td>
<td>43</td>
<td>2C</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>B6</td>
<td>9C</td>
<td>EC</td>
<td>64</td>
<td>44</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>B7</td>
<td>9F</td>
<td>EB</td>
<td>65</td>
<td>45</td>
<td>09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>B8</td>
<td>AF</td>
<td>DF</td>
<td>66</td>
<td>46</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>B9</td>
<td>51</td>
<td>A0</td>
<td>67</td>
<td>47</td>
<td>EA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC</td>
<td>8D</td>
<td>20</td>
<td>68</td>
<td>48</td>
<td>CF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>9D</td>
<td>73</td>
<td>21</td>
<td>75</td>
<td>52</td>
<td>EF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>8E</td>
<td>9A</td>
<td>E0</td>
<td>D0</td>
<td>C0</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>FF</td>
<td>CE</td>
<td>EE</td>
<td>8A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix 4.

System defined parameter values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of header lines</td>
<td>3 (GH3)</td>
</tr>
<tr>
<td>Page number</td>
<td>1 (G#1)</td>
</tr>
<tr>
<td>Page title</td>
<td>51 blanks date 10 blanks</td>
</tr>
<tr>
<td>Page no. follower title</td>
<td>HA-T</td>
</tr>
<tr>
<td>Record no. left margin</td>
<td>90 (GR90)</td>
</tr>
<tr>
<td>Figure spacings</td>
<td>G11 G21 G32</td>
</tr>
<tr>
<td>Footnote size</td>
<td>20 (GN20)</td>
</tr>
<tr>
<td>Footnote markers</td>
<td>H12345T</td>
</tr>
<tr>
<td>Page depth</td>
<td>58 (GD58)</td>
</tr>
<tr>
<td>Line spacing</td>
<td>1 (GS1)</td>
</tr>
<tr>
<td>Current indentation</td>
<td>N (IN)</td>
</tr>
</tbody>
</table>

System defined descriptors

<table>
<thead>
<tr>
<th>char</th>
<th>string</th>
<th>defining command</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>$KB-('-'')</td>
</tr>
<tr>
<td>;</td>
<td></td>
<td>$KB,('')</td>
</tr>
</tbody>
</table>

Table of the system defined word break descriptors.

<table>
<thead>
<tr>
<th>char</th>
<th>string</th>
<th>size</th>
<th>defining command</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>.</td>
<td>69</td>
<td>%KFi('.', 69)</td>
</tr>
</tbody>
</table>

Table of the system defined word or line fill descriptor.
<table>
<thead>
<tr>
<th>char</th>
<th>string</th>
<th>assoc</th>
<th>defining command</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>_</td>
<td>C</td>
<td>$%KUn('_','C')$</td>
</tr>
<tr>
<td>d</td>
<td>_</td>
<td>C</td>
<td>$%K Ud('$$EB','C')$</td>
</tr>
<tr>
<td>b</td>
<td>~</td>
<td>N</td>
<td>$%KUb('$$EC','N')$</td>
</tr>
</tbody>
</table>

Table of the system defined underscore descriptors.

<table>
<thead>
<tr>
<th>char</th>
<th>string</th>
<th>assoc</th>
<th>defining command</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>.</td>
<td>C</td>
<td>$%K Oa('$$OF','C')$</td>
</tr>
<tr>
<td>g</td>
<td>_</td>
<td>C</td>
<td>$%K Og('$$2B','C')$</td>
</tr>
<tr>
<td>f</td>
<td>.</td>
<td>C</td>
<td>$%K If('$$08','C')$</td>
</tr>
<tr>
<td>t</td>
<td>_</td>
<td>C</td>
<td>$%K Ot('$$29','C')$</td>
</tr>
<tr>
<td>u</td>
<td>_</td>
<td>C</td>
<td>$%K Ou('$$11','C')$</td>
</tr>
<tr>
<td>c</td>
<td>.</td>
<td>C</td>
<td>$%K Oc('$$09','C')$</td>
</tr>
<tr>
<td>s</td>
<td>/</td>
<td>C</td>
<td>$%K Os('$$/','C')$</td>
</tr>
</tbody>
</table>

Table of the system defined constant overprint descriptors.

<table>
<thead>
<tr>
<th>parameter</th>
<th>indentation descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>pw</td>
<td>0</td>
</tr>
<tr>
<td>ir</td>
<td>0</td>
</tr>
<tr>
<td>rm</td>
<td>68</td>
</tr>
<tr>
<td>me</td>
<td>4</td>
</tr>
<tr>
<td>vsf</td>
<td>4</td>
</tr>
<tr>
<td>lmf</td>
<td>4</td>
</tr>
<tr>
<td>ssf</td>
<td>0</td>
</tr>
<tr>
<td>thf</td>
<td>18</td>
</tr>
<tr>
<td>vsp</td>
<td>1</td>
</tr>
<tr>
<td>lmp</td>
<td>14</td>
</tr>
<tr>
<td>ssmp</td>
<td>0</td>
</tr>
<tr>
<td>tbp</td>
<td>5</td>
</tr>
<tr>
<td>vsc</td>
<td>0</td>
</tr>
<tr>
<td>lmc</td>
<td>9</td>
</tr>
<tr>
<td>ssc</td>
<td>0</td>
</tr>
</tbody>
</table>

$\%KIN(0,0,68,4,4,4,0,18,1,14,0,5,0,9,0)$
$\%KIR(1,13,68,4,1,15,0,5,1,20,0,5,0,15,0)$
$\%KIL(1,16,68,4,1,18,0,5,1,23,0,5,0,18,0)$
$\%KID(1,19,68,4,1,21,0,5,1,26,0,5,0,21,0)$
$\%KIDEP(2,17,68,4,1,18,0,5,1,23,0,5,0,18,0)$
$\%KIFOOT(1,17,68,4,0,0,8,0,0,0,0,13,0,0,0,0)$
$\%KIFIG(0,0,63,4,0,0,6,0,0,0,16,0,0,0,11,0)$

Table of the system defined indentation descriptors.
A text formatter, version 2

Definition of list nodes.

* List of words to be placed in one line

1  DATA('WRDLST(WRDSTR,WRDOVR,NXTWRD)')
*
* The fields are:
  WRDLSTHD - the character string of the word
  WRDOVR - pointer to the list of overprint strings per word
  NXTWRD - pointer to the next word in this list
*
* The associated variables are:
  WRDLSTHD - Pointer to the header of the wordlist. Must be
  properly initialized at start. Remains unchanged.
  CURWRD - pointer to the last word entry in the word list.
  Must be set to WRDLSTHD when initiating a new line.
  WRDCNT - Counts the number of words in the word list. Initialized
  to 0 when a new line is started.
  LINLEN - the character column position of the first character
  of the word, if the line is built single spaced.
  Must be initialized to LEFT MARGIN when starting new lin
  WRD - the string of the word being currently built.
  Must be initialized to NULL when starting a new word.

* List of overprint strings per word

2  DATA('USCLST(USCSTR,USCPOS,USCASC,NXTUSC)')
*
* The fields are:
  USCSTR - the character string to overprint
  USCPOS - the starting position of the string relative to the
  beginning of the word
  USCASC - the line with which the overprint is to associated,
  i.e. 'C' for current line and 'N' for next line
  NXTUSC - a pointer to the next element in this list
*
* The associated variables are:
  USCLSTHD - Pointer to the top of the overprint string stack
  of the current word. Must be NULL for new word.
  USCLSTPT - pointer to the USCLST node for which the underscore
  must still be completed.
  USCLFTG - flag telling whether underscoring is in progress.
  0 - not in progress. 1 - in progress not continuous.
  2 - in progress and continuous.
  SEPLSTHD - pointer to the word separator entry stack. It must
  be NULL when starting on a new word.

* List of lines to be printed

3  DATA('LINLST(LINSTE,LINSPC,LINOVR,LINFOT,NXTLIN)')
*
* The fields are:
  LINSTR - the character string of the line to be printed
  LINSPC - the number of blank lines to precede this line
  LINOVR - pointer to the list of overprint lines
  LINFOT - pointer to a footnote descriptor node. This
  node is one of type FIGLST. The footnote is associated
  with this next line in the list.
  NXTLIN - pointer to the next line in this list
*
* The associated variables are:
  LINLSTHD - pointer to the first title line of the page list. It
  remains unchanged throughout the execution.
  CURLIN - pointer to the last line in the page line list. It must
  be set to PAGHAD whenever a new page is started.
  LINCNT - total number (heading lines and all) of physical print
  lines in the page. Must be set to PAGINI for new page.
Definition of list nodes.

* PAGHAD - pointer to the last line of the page header line list.
  It is modified only by 'FGR' commands.
* PAGINI - counter of the physical lines in the page header list.
  It is modified only by 'FGR' commands.
* FOTLSTHD - pointer to the header of the footnote list. Remains unchanged.
* CURFOT - pointer to the last footnote list line.

---------
* List of overprint lines per line
  
4   DATA('OVRLST(OVRLST,OVRASC,NXTOVR)')
* The fields are:
  OVRSTR - the character string of the line to be overprinted
  OVRASC - the line association for the overprint (see USCASC)
  NXTOVR - pointer to the next overprint line in this list
* The associated variable is:
  OVRLSTHD - pointer to the top of the overprint line stack.

---------
* Figure list entry definition

5   DATA('FIGLIST(FIGBEG,FIGEND,FIGCNT,NXTFIG)')
* The fields are:
  FIGBEG - pointer to the first line in the figure list.
  FIGEND - pointer to the last line in the figure list.
  FIGCNT - number of lines in the figure list.
  NXTFIG - pointer to the next entry in the figure list.
* The associated variables are:
  FIGLISTHD - pointer to the figure list header node. Remains unchanged.
  CURFIG - pointer to the last figure entry. Must be FIGLISTHD if no figure exists.
  LINFOT() - associates next line with footnote list.
  FIGIND - pointer to the figure descriptor just built
  FCTIND - pointer to the footnote list descriptor just built
  FIGPOT() - function returning a figure or footnote descriptor

---------
* Underscore and sign overprint descriptor

6   DATA('ULNDDEF(ULNSYM,ULNASC)')
* The fields are:
  ULNSYM - the character to be used when overprinting
  ULNASC - the line association of this descriptor
* The associated variables are:
  ULNPTR - pointer to the underscore descriptor currently in use.
  ZUSC char - pointer to the underscore descriptor 'char'.
  SINPTR - pointer to the current single overprint descriptor in use.
  ZSIN char - pointer to the single overprint descriptor 'char'.

---------
* Indentation block entry definition

7   DATA('INDDEF(INTCNT,PRWRGT,RTMRG,BKEND,SPACF,LEFTF,MNLF,TXMF,' + 'SPACF,LEFTF,MNLF,TXMF,SPACC,LEFTC,MNLC)')
* The fields are:
A text formatter, version 2

Definition of list nodes.

* INTCNT = number of primal words for the first line.
* PRMRGT = primal word right margin for first line.
* RGMTBG = right margin of the block
* BLKEND = number of lines at end of block which have to remain
  together on a page.
* SPAC (F,P or C) = number of blank lines to precede each of the
  three line types.
* LEFT (F,P or C) = left margin of each of the three line types.
* MINL (F,P or C) = minimum single spaced left string length of
  each of the three line types.
* TXXM (F or P) = the minimum number of lines of the beginning of
  a new text block, which have to remain together on a page.

* The associated variables are:
  * CURIDN = pointer to the current indentation block descriptor.
  * ZIDN nam = pointer to the indentation block descriptor 'nam'.
A text formatter, version 2

Read cards, break them into words, initiate text commands.
* Define the reading patterns. These patterns fail only if the rest of
* the card is blank.
* 8 FINDA = (SPAN(' ')) | ' ' FENCE BREAK(' ' $P') . FOUNDWRD
9 FINDB = BREAK(' $P') . FOUNDWRD
* The input variables are:
* WRD - the word being formed.
* LINLEN - the length of the current line plus one if single
  spaced lines where built.
* LINSIZ - the character position of the right hand margin.
* CURWRD - pointer to the last word inserted into word list.
* WRDLSTHD - pointer to word list header.
* WRCNT - number of entries in the word list
* CRDEND - the termination and paragraphing flag - serves to
  distinguish between formatted and unformatted mode.
* USCLSTHD - pointer to the stack of word overprint strings.
* SEPLSTHD - pointer to the separator point stack. This stack is
  ordered from the rightmost to the leftmost separator
  when traversing the stack from top to bottom.
* Define the function header
10 DEFINE('FINDER()') ;* read on entry point
11 DEFINE(READER()) :END(READER)
* Read a card and create a paragraph command if first column is blank
* 12 READER INPUT ETAB(S) . CARD REM . SEQN :F(RETURN)
13 CARD CRDEND = ' $P' ;* Create paragraph command
14 CARD = CARD CRDEND ;* Guarantee word break
* Find a word segment, i.e. a string delimited by an escape
* 15 FINDER CARD FIND = :F(READER)
16 WRD = WRD FOUNDWRD ;* FOUNDWRD is generated by FIND
17 CARD '$P' = :F(PUTWRD)
18 WRD = WRD ESCAPEFN(CENCND) :F(PUTWRD)
19 CARD IDENT(FINDA,FIND)' ' = :F(FINDER)
* Verify if the word found overflows line
* 20 PUTWRD TEMP1 = DIFFER(WRD,' ') LINLEN + SIZE(WRD) :F(ENDWORD)
21 LINLEN = LE(TEMP1,LINSIZ) TEMP1 + 1 :S(ACCNWRD)
* Find a word separation if any is available
* 22 LINR1 DIFFER(SEPLSTHD,'') :F(PUTWRD2)
23 TEMP1 = USCPGS(SEPLSTHD)
24 TEMP2 = TEMP1 + LINLEN + SIZE(USCSTR(SEPLSTHD))
25 SEPLSTHD = GT(TEMP2,LINSIZ) NXTUSC(SEPLSTHD) :S(LINR1)
* A valid separation exists. Complete any pending underscore.
* 26 DIFFER(USCLSTPT,'') :F(LINR2)
27 USCSTR(USCLSTPT) = DUPL(USCSTR(USCLSTPT), SIZE(WRD) -
  + USCPGS(USCLSTPT))
* Break the word into two portions
* 28 LINR2 LINLEN = TEMP2 + 1 ;* there is a space after
29 WRCNT = WRCNT + 1 ;* account word
30 NXTWRD(CURWRD) = WRLST(' ')
31 CURWRD = NXTWRD(CURWRD)
32 WRD LEN(TEMP1) . WRDSTR(CURWRD) = ;* break word
33 WRDSTR(CURWRD) = WRDSTR(CURWRD) USCSTR(SEPLSTHD)
* Initialize to break overprints
A text formatter, version 2

Read cards, break them into words, initiate text commands.

*  
34  TEMP2 = USCLSTHD
35  TEMP3 = ;* next word overprints
36  TEMP4 = ;* this word overprints
  
  * Loop through all overprints and break them if needed
  *
37  LINR4  DIFFER(TEMP2,"") ;F(LINR5)
38     TEMP3 = GT(USCPOS(TEMP2),TEMP1)  
       + USCLST(USCSTR(TEMP2),USCPOS(TEMP2) - TEMP1,
       + USCASC(TEMP2),TEMP3) ;S(LINR3)
39     TEMP4 = LE(USCPOS(TEMP2) + SIZE(USCSTR(TEMP2)),TEMP1)  
       + USCLST(USCSTR(TEMP2),USCPOS(TEMP2),
       + USCASC(TEMP2),TEMP4) ;S(LINR3)
  
  * Overprint string must be broken
  *
40     TEMP3 = USCLST('"',0,USCASC(TEMP2),TEMP3)
41     TEMP4 = USCLST('"',USCPOS(TEMP2),USCASC(TEMP2),TEMP4)
42     USCSTR(TEMP2) LEN(TEMP1) . USCSTR(TEMP4) REM . USCSTR(TEMP3)
43  LINR3  TEMP2 = NXTUSC(TEMP2) :(LINR4)
  
  * The word and all its overprints have been broken line it
  *
44  LINR5  WRDDEVR(CURWRD) = TEMP4
45  USCLSTHD = TEMP3
46     LFTMIN = LINER(LFTMIN) :(PUTWRD1)
  
  * Line has been filled to exaustion. Generate a text line.
  *
47  PUTWRD2  LFTMIN = LINER(LFTMIN) ;F(ACCNWRD)
48  PUTWRD1  LINLEN = LINLEN + SIZE(WRD) + 1
  
  * Account this word in the line
  *
49  ACCNWRD  WRDCNT = WRD + 1
50     NXTWRD(CURWRD) = WRDLST(WRD,USCLSTHD)
51     CURWRD = NXTWRD(CURWRD)
  
  * Verify if end of word escape requested
  *
52  ENDWORD  USCLSTHD = ;* clear overprints of word
53     DIFFER(ESCAPE,"") ;S(ESCNW)
54  ESCAPRET  WRD = ;(FINDER)
A text formatter, version 2

Escape handling within READER()

* The used variables are:
  * ESCAPE - a list of single character escape commands.
  * ESCTEMP - a list of single character escape commands which is
    to be followed after the next word is completed.
  *
* Distribute the escape commands
*
55 ESCNDN ESCTEMP = ;* initialize result string
56 ESCAPEND ESCAPE LEN(1), II = :S($('ESCAP' II))
57 ESCAPE = ESCTEMP ;(ESCAPRED)

*ESCAPEu - see underscore command handling in overprint functions.
*
* Primal word handling.
* Initialize parameters. TEMP1 := NULL if WRD=""
*
58 ESCAPP TEMP1 = NXTWED(WRLSTHD)
59 DIFFER(WRD,"") :F(PRIM4)
60 DIFFER(TEMP1, CURRD) :F(PRIM3)

* Combine 2 words into one single word
*
61 TEMP2 = SIZE(WRDSTR(TEMP1)) + 1
62 WRDSTR(TEMP1) = WRDSTR(TEMP1) = ' ' WRDSTR(CURRD)

* Correct USCPNPOS entries of all overprint list entries of 2nd word
*
63 TEMP3 = WRDSTR(CURRD)
64 DIFFER(TEMP3,"") :F(PRIM2)
65 PRIM1 USCPNPOS(TEMP3) = USCPNPOS(TEMP3) + TEMP2
66 TEMP3 = DIFFER(NXTUSC(TEMP3),"") NXTUSC(TEMP3) :S(PRIM1)

* Insert overprint list of 2nd word preceding that of 1st word
*
67 NXTUSC(TEMP3) = WRDOVR(TEMP1)
68 WRDOVR(TEMP1) = WRDOVR(CURRD)

* Reset word list pointers to one only word so far
*
69 PRIM2 CURRD = TEMP1
70 NXRWD(TEMP1) =

* Count primal words and test if end
*
71 PRIM3 PRMCNT = PRMCNT - 1
72 PRIM4 ESCTEMP = GT(PRMCNT, 0) ESCTEMP 'p' :S(ESCAPEND)
73 LINSIZ = RGTMRG(CURIDN)
74 WRDCTNP = DIFFER(TEMP1,"") 1 :F(ESCAPEND)
75 LINLEN = LINLEN - SIZE(WRDSTR(CURRD)) - 1 : (ESCAPEND)

* Paragraph escape handling 'P'
*
77 ESCAPP TERMIN('P') ;(ESCAPEND)

* Indentation command escape handling 'INAM(bk|')'
* Complete line and finish previous block if necessary
*
77 ESCAP1 TERMIN('F') ;* complete line
* obtain indentation descriptor
*
78 CARD BREAK(' ', II) = DIFFER($('ZIDN' II), "") :F(ESCAPEND)
79 CURIDN = $(ZIDN II) ;* current indentation descr.
A text formatter, version 2

Escape handling within READER()

```
80 ESCTMP = INDENT() : (ESCAPEND)
*
* Skip nn lines command escape handling "%Z(N|L)(I)(R)nn"
*
81 ESCAPZ CARD "N" = TERMIN("P")
82 CARD "L" = : F(ESCAPZ1)
83 LINER(LFTMIN) : (ESCAPZ2)
84 ESCAPZ1 LINER(1000) ;* done if not new block
85 ESCAPZ2 LNPARSRT("C") +
86 CARD ("I" | "") * I1 ("R" | ") * I2 ;* SPAN("0123456789") * SCLIN =
87 SPCLIN = DIFFER(I2,"") SPCLIN + SPCLING ;* F(ESCAPEND)
88 DIFFER(I1,"") : F(ESCAPEND)
*
* Construct unconditional space
*
89 NXTWD(WRDLSTHD) = WRDLST(" ")
90 LINER(1000) : (ESCAPEND)
*
* Definition of a new set of page header lines "%GH(n)"
*
91 ESCAPG TERMIN("P")
92 LNPARSRT("C")
93 DIFFER(CURLIN,PAGHD) PAGER(LINCNT)
*
* Obtain number of page heading spaces
*
94 CARD ANY("0123456789") * TEMP2 = : F(ESCAPEND)
95 PAGINI = TEMP2
96 LINCNT = PAGINI ;* reset page line count
97 PAGAV = PAGINI
98 PAGHD = LINLSTHD
99 TEMP1 =
100 PAGHD = GT(PAGINI,0) LINLST(" ,1,"",","",") ;* F(ESCAPG2)
*
* Create PAGINI-2 lines
*
101 TEMP1 = PAGHD
102 ESCAPG1 TEMP2 = GT(TEMP2,1) TEMP2 - 1 ;* F(ESCAPG2)
103 TEMP1 = LINLST(" ,1,"",","",", TEMP1) ;* (ESCAPG1)
*
* Put created list into page header list
*
104 ESCAPG2 NXTLIN(LINLSTHD) = TEMP1
105 CURLIN = PAGHD : (ESCAPEND)
*
* Termination command escape handling "%T(N|bk)"
*
106 ESCAPT CARD ("N" *TERMIN("P") | ") =
107 LINER(1000) ;* done if not end of block
108 LNPARSRT("C") ;* set to continuation line
109 FNSTACK LENC(1) = : F(ESCAPEND)\$S(RETURN)
*
* Unformatted mode escape handling "%U(N)"
* Terminate line and block if required
*
110 ESCAPU CARD "N" = TERMIN("P") ;* end of block too
111 LINER(1000) ;* done if not end of block
112 LNPARSRT("C") ;* by defn continuation line
*
* Set unformatted mode parameters
*
113 FIND = FINDB ;* no blanks as delimiters
114 CRDEND = "f/" ;* force end of word
A text formatter, version 2

Escape handling within READER()

115 WRD = FNSTACK = 'U' FNSTACK ;* erase contents of word
116 READER( );* $T$ may proceed
117 ;* read from new card
118 FIND = FINDA ;* blanks are delimiters again
119 CRDEND = ' ' ;* normal end of card and parag
120 CARD RTAB(2) * I1 "$/$" = I1 "$" ;* (ESCAPEND)

*----------------------------------------------------------
* End of card and end of line command "$/$"
*----------------------------------------------------------
121 ESCAP/ LINER(1000)
122 LINPARSET('C')
123CARD = ;* (ESCAPEND)

*----------------------------------------------------------
* Footnote referencing word has been accounted. Line is next one "$N$"
*----------------------------------------------------------
124 ESCAPN LINCNT = LINCNT + FIGCNT(FOTIND)
125 LINFOT(CURLIN) = FOTIND ;* footnote to previous line
126 ESCAPN1 FOTIND = ;* (ESCAPEND)

*----------------------------------------------------------
* Word has been accounted. Erase separator record
*----------------------------------------------------------
127 ESCAPs SEPLSTHD = ;* (ESCAPEND)
128 END.READER
A text formatter, version 2

Generate a text line from a word list.

* The input variables are:
  * WRDLSTHD, WRDCNT, LINLEN, CURWRD, LINSIZ as in READER()
  * LFTMRG – the left margin of non primal words
  * INTRGT – the right margin of the primal words
  * LFTMIN – the minimum string length of the single spaced portion of the line being built.
  * SPCLIN – the number of blank lines to precede the line being built
  * LINLSTHD – pointer to the header of the line list of the current text block being built.
  * LINCNT – the number of physical lines in this block
  * CURLIN – pointer to the last line in this block
  * OVRLSTHD – pointer to the top of the overprint line stack of line being formed
  * CURIDN – pointer to the current indentation block descriptor
  * SPCING – number of blank lines to provide between two lines if not overridden by a spacing text command.

129 DEFINE ('LINER(LFTMIN)') : (END.LINER)

* Initialize

130 LINER
   CURWRD = DIFFER(NXTWRD(WRDLSTHD), '') +
   NXTWRD(WRDLSTHD) :F (RETURN)
131 LIN = '' DUPL( '', LFTMRG) ;* Beginning of line is blank
   * Put the intermediate right margin in if there is any.
   * GT(INTRGT, 0) ;F (INLEFT)
   LIN = '' DUPL( '', INTRGT - SIZE(WRDLSTWRD(CURWRD))) +
   DIFFER(WRDOVR(CURWRD)) OVERLAYER(CURWRD)
134 LIN = LIN DUPL( '', LFTMRG - INTRGT)
135 CURWRD = NXTWRD(CURWRD)
136 WRDCNT = WRDCNT - 1 ;* one word is in already
137 INTRGT = 0 ;* inhibit further primal lines
138 ESCAPE BREAK('p') . If 'p' = 11
   * Put the initial single spaced section of the line in
   * INLEFT DIFFER(CURWRD, '') ;F (PUTLIN)
140 LIN = LIN WRDLSTWRD(CURWRD)
141 DIFFER(WRDOVR(CURWRD), '') OVERLAYER(CURWRD)
143 CURWRD = NXTWRD(CURWRD)
144 WRDCNT = WRDCNT - 1
   * Test if initial portion of line has been exhausted
   * LIN = LT(SIZE(LIN), LFTMIN) LIN + ' ' :S (INLEFT)
145 DIFFER(CURWRD, '') ;F (PUTLIN)
   * Compute space factor for weighted extra blank insertion
   * SPFCT = (100 * (LINSIZ - LINLEN + 1)) / WRDCNT + 100
147 OLDSPC = 0
148 WRDCNT = 1
   * Put word in line preceding it by sufficient spaces
   * ALLWRDS
150 NEWSPC = (WRDCNT * SPFCT + 52) / 100
151 LIN = LIN DUPL( '', NEWSPC - OLDSPC) WRDLSTWRD(CURWRD)
152 DIFFER(WRDOVR(CURWRD), '') OVERLAYER(CURWRD)
   * Account inserted word
153 OLDSPC = NEWSPC
154 CURWRD = NXTWRD(CURWRD)
A text formatter, version 2

Generate a text line from a word list.

155 WRDCNT = DIFFER(CURWRD,'a') WRDCNT + 1 :S(ALLWRDS)
*  
* Line building has been completed. Put line into page list
*  
156 PUTF01 DIFFER(MINLIN,'a') DIFFER(MINLIN,'A') :F(PUTL0)
*  
* It is the first line of a text block. Test if block fits
*  
157 PUTF03 GT(LINCNT + MINLIN,PAGSIZ) :F(PUTL2)
158 LE(LINCNT,PAGSIZ) PAGER(LINCNT) :S(PUTL3)
159 PAGER(LINCNT - MINEND) :{(PUTL3)
160 PUTF02 MINLIN = :{(PUTL0)
*  
* Introduce the record number into the line
*  
161 PUTF00 LIN = DIFFER(CRDEND,'aV') LIN DUPL(1,SEQALN -
* SIZE(LIN)) SEQN
*  
* Account this line in the line list
*  
162 PUTF1 SPCLIN = IDENT(CURLIN,PAGHAD) 0 ;* no space on top
163 LINCNT = LINCNT + SPCLIN + 1
164 NXTLIN(CURLIN) = LINLST(LIN,SPCLIN + 1,OVRLSTHD)
165 CURLIN = NXTLIN(CURLIN)
166 OVRLSTHD =
*  
* Reset the line alignment governing parameters to continuation line
*  
167 SPCLIN = SPACC(CURIDN) + SPCING
168 LFTMRG = LEFTC(CURIDN)
169 LINER = MINLC(CURIDN) + LFTMRG
*  
* Reset word list pointers and counters
*  
170 CURWRD = WRDLSTHD
171 NXTWRD(CURWRD) =
172 WRDCNT = 0
173 LINLEN = LFTMRG
*  
* Is this end of page?
*  
174 GT(LINCNT - MINEND,PAGSIZ) PAGER(PAGSIZ) :{(RETURN)
175 END•LINER
A text formatter, version 2

Auxiliary functions.

* * Reset the line and block governing parameters
* CURIDN - pointer to the current indentation descriptor
* SPCING - number of blank lines between two consecutive lines
* 176 DEFINE('LINPARSET(TMP)') (:END.LINPARSET)
* no further primal words
* LINSIZ = RGTMRG(CURIDN)
* 179 ESCTMP = ;* guarantee no escape after
* Set line parameters
* 180 SPCLIN = EVAL('SPAC' TMP '(CURIDN)') + SPCING
181 LFTMRG = EVAL('LEFT' TMP '(CURIDN)')
182 LFTMIN = EVAL('MINI' TMP '(CURIDN)') + LFTMRG
183 LINLEN = LFTMRG
184 IDENT(TMF,'C') :S(RETURN)
* Set block parameters
* 185 MINEND = BLKEND(CURIDN)
186 MINLIN = EVAL('TTM1' TMP '(CURIDN)')
187 MINEND = IDENT(MINLIN,'A') PAGSIZ - PAGINI (:RETURN)
188 END.LINPARSET
* Convert a hexadecimal character to decimal value
* 189 DEFINE('CONV(TMP)') (:END.CONV)
190 CONV TMP ANY('0123456789') . CONV :S(RETURN)
191 CONV = REPLACE(TMP,'ABCDEF','012345') + 10 (:RETURN)
192 END.CONV
* Build a string until $T$ is found
* 193 DEFINE('BUILDSSTR )INTRGT,PRMCNT,LINSIZ,MINEND,SPCLIN,LFTMRG,'
+ 'LFTMIN,MINSLIN,FIND,ESCAPE,CRDSAV,WRDLSTHD,CURWRD,WRDCNT,'
+ 'LINLEN,WRD,USCLSTHD,USCLSTPT,USCFGL,USCFPR,LINLSTHD','
+ 'CURLIN,LINCNT,PAGSIZ,LINSAV,FOTIND,SPCING,CENCMD,II,12')
+ (:END.BUILDSSTR)
* Inhibit line building during string building
* 194 BUILDSSTR LINLEN = 0 ;* set fake line parameters
195 LFTMIN = 0
196 LFTMIN = 1000
197 LINSIZ = 1000
198 LINCNT = 0
* Set reading pattern and inhibit cent functions
* 199 FIND = FINDB ;* unformatted string
200 CENCMD = 'VTJ#&'
201 CRDSAV = CRDEND
202 CARD RTAB(SIZE(CRDEND)) . II CRDEND = II 'XY'
203 CRDEND = 'XY'
* Create list headers needed to properly function
* 204 CURWRD = WRDLST('')
205 WRDLSTHD = CURWRD
206 CURLIN = LINLST('') ;* guarantee headers
207 LINLSTHD = CURLIN
A text formatter, version 2

Auxiliary functions:

```
208     PAGSIZ = 1000 ;* allow no printing activity
209     FNSTACK = 'B' FNSTACK ;* allow 'T' to return
210     FIND( ) ;* go and readon

* The return is achieved by 'T'. the line has been built

211     CRDEND = CRDSAV
212     CARD RTAB(2) * I1 'V' = I1 CRDEND
213     BUILTSTR = CURLIN ;(RETURN)
```

```
214 END.BUILTSTR
```

```
* Terminate a block with a paragraph command

* DEFINE('TERMIN(TEMP1)') ;(END.TERMIN)

215     TERMIN     LINER(1000) ;* force end of line

* Test if complete block goes on one page

216     IDENT(MINLIN,'A') GT(LINCNT,PAGSIZ) PAGER(LINSAV)

* Redefine line and page parameters

217     LINSAY = LINCNT ;* foresee case no paging
218     DIFFER(TEMP1,'F') LPARSET('F') ;(RETURN)
```

```
219 END.TERMIN
```

```
* Set all parameters for a new indentation block

* DEFINE('INDENT()') ;(END.INDENT)

220     INDENT     LPARSET('F') ;* block and line parameters

* Set constant indentation block parameters

221     INTRGT = PRMGR(G(RUIRDN)) ;* primal right margin
222     PRMCNT = INTCUNT(URIKN) ;* no. of primal words
223     LINSIZ = RECTMRG(URIKN)

* Prepare primal words parameters

224     INDENT = GT(PRMCNT,0) 'P' ;F(RETURN)
225     LINSIZE = 1000 ;(RETURN)
```

```
226 END.INDENT
```

```
* Collect footnote and figure text. This function serves only to save
* and initialize values

* DEFINE('FIGFOT(URIKD,INTRGT,PRMCNT,LINSIZ,MINEND,SPCLN,LFTMRG,' +
  'LFTMIN,MINLINFIND,CRDEND,ESCAPE,WRDLSTDH,CURWORD,WRDCNT,,' +
  'LINLEN,WRC,USDLSTDH,USDLSTPP,USCLFLG,USCPTR,USCLSTDH,' +
  'CURLIN,LINCNT,PAGSIZ,LINSAV,POTIND,SPCLNG,CRDSAV')

227     FIGFOT     PAGSIZ = 1000 ;* inhibit paging
228     WRDLSTDH = WRDLST( )
229     CURWORD = WRDLSTDH
230     LINLSTDH = LINLSTDH
231     CURLIN = LINLSTDH
232     ESCAPE = INDENT( ) ;* set parameters of line
```

```
* Set remaining parameters

233     FIND = FINDA ;* blanks delimit
234     CRDEND = ' ' ;* normal paragraph
235     FNSTACK = 'N' FNSTACK ;* allow 'T' to operate
```

236     FIND( )
```
A text formatter, version 2

Auxiliary functions.

* The figure and footnote have been completed reset
*  
242    FIGFOT = FIGLST(NXILIN(LINLSTHD), CURLIN, LINCNT) : (RETURN)
243    END. FIGFOT
A text formatter, version 2

Generate overprint lines from overprint list.

* The input parameters are:
  * WRD = as in READER() and LINER()
  * LIN = the current line being formed by LINER()
  * OVRLSTHD = pointer to the top of the overprint line stack
  * of the line currently being formed by LINER()

244 DEFINE('OVERLAYER(WRD)') : (END.OVERLAYER)

* Initialize

245 OVERLAYER POS = SIZE(LIN) - SIZE(WRDSTR(WRD))
246 WRD = WRD0VR(WRD) ;* enter overprint string start

* Loop through all overprint strings of the word

247 OVRLLOOP2 DIFFER(WRD,'') : F(RETURN)
248 TMP = POS + USPCSC(WRD)

* Loop through overprint lines in search for one where string fits

249 OVRLLOOP1 DIFFER(PTR,'') : F(OVRLLOOP5)

* This line is valid iff line associations are equal and field is blank

251 OVRLSTHD IDENT(OVRASC(PTR),USCASC(WRD)) LEN(TMP) . I1 +
  DUPL(' ','SIZE(USCSTR(WRD)))
  = I1 USCSTR(WRD) : S(OVRLLOOP4)

* get next line. If lines are exhausted, build a new line.

252 OVRLLOOP5 OVRLSTHD = OVRLST('+' DUPL(' ','TMP - 1) USCSTR(WRD) +
  DUPL(' ','120 - TMP - SIZE(USCSTR(WRD)))),
  USCASC(WRD),OVRLSTHD

* Get next overprint string in word

254 OVRLCNP4 WRD = NXTUSC(WRD) : (OVRLLOOP2)
255 END.OVERLAYER
A text formatter, version 2

Output a page of lines.

* The input variables are:
  * BLKLSTHD, CURBLK, PAGCNT, PAGSIZ as in LINER()
  * PAGENO - the number of the current page
  * TITLE2 - the title portion to follow the page number
  * PAGINI - the number of lines in the page header
  * PAGHDR - pointer to the last line in the page header
  * FOTLSTHD - pointer to the header of the footnote line list
  * PNDFOTHD - pointer to the first line of a continuation
    footnote
  * PNDCURFOT - pointer to the last line in the pending footnote
    line list.
  * PNDCNT - number of lines in the pending footnote line list
  * FIGLSTHD - pointer to the header of the figure list
  * CURFIG - pointer to the last figure in the figure list

256 DEFINE('PAGER(CNT)TEMP1,TEMP3,TEMP4') : (END.PAGER)

* Output title line and count pages

257 PAGER OUTPUT = LINSTR(LINNLSTHD) PAGENO TITLE2
258     PAGENO = DIFFER(PAGENO,"" ) PAGENC + 1
*
* Get ready to print CNT lines
*
259 NXTLIN(FOTLSTHD) = ; * no footnote on this page
260 CURFO = FOTLSTHD
261 TEMP4 =
262 LINCNT = LINCNT - CNT
263 PRINT(LINNLSTHD) : S(PAGER1)
*
* The list has been exhausted.
*
264 CURLIN = PAGHDR
265     LINCNT = PAGINI + CNT : (PAGER2)
*
* There are still lines to be printed in the list
*
266 PAGER1 LINCNT = LINCNT + CNT + PAGINI - LINSPC(TEMP2) + 1
267 LINSPC(TEMP2) = 1
268 PAGER2 NXTLIN(PAGHDR) = TEMP2 ; * delete printed lines
*
* Prepare to print all the footnote lines
*
269 CNT = 1000
270 PRINT(LINNLSTHD) ; * print collected footnotes
271 NXTLIN(FOTLSTHD) =
272 FOTCOT = 0
273 DIFFER(TEMP4,"" ) PENDER(""
*
* Initialize pointers to figure and figure insertion
*
274 TEMP1 = DIFFER(NXTFIG(FIGLSTHD)) NXTFIG(FIGLSTHD)
275 + : F(PAGER9)
276 TEMP3 = LINCNT - PAGINI ; * figure goes on top
277 TEMP4 = PAGHDR ; * top of page
278 NXTFIG(FIGLSTHD) = ; * avoid recursive figures
279 FIGBEG(TEMP1) = LINLST(FIGTIT,0,,\, FIGBEG(TEMP1))
*
* Insert the figure into the page. TEMP4 points to last line
*
280 PAGER4 NXTLIN FIGEND(TEMP1)) = NXTLIN(TEMP4) ; * put continuation
281 NXTLIN(TEMP4) = FIGBEG(TEMP1) ; * figure is in now
282 CURLIN = IDENT(CURLIN,TEMP4) FIGEND(TEMP1)
283 TEMP4 = FIGEND(TEMP1) ; * last line = TEMP4
284 LINCNT = LINCNT + FIGCNT(TEMP1) ; * account space
*
* If now the text overflows the page, then print as many pages as need
A text formatter, version 2

Output a page of lines.

* 284 PAGER5 GT(LINCNT - TEMP3,PAGSIZ) PAGER(PAGSIZ) :S(PAGER5)
  * The line TEMP4 has not been output due to GT relation. Get next figure
  *
  285 TEMP1 = NXTFIG(IEMP1)
  286 CURFIG = IDENT(TEMP1) FIGLSTHD :S(PAGER6)
  287 LINCNT + FIGCNT(TEMP1) - TEMP3,PAGSIZ) :S(PAGER4)
  *
  * No more figures can be placed. Set minimum after figure spacing
  *
  288 PAGER6 IDENT(NXTLIN(TEMP4)) :S(PAGER7)
  289 TEMP2 = LINSFC(NXTLIN(TEMP4))
  290 LINCNT = LT(TEMP2,FIGSP3) LINCNT - TEMP2 + FIGSP3
  :F(PAGER8)
  *
  291 LINSFC(NXTLIN(TEMP4)) = FIGSP3 :S(PAGER8)
  292 PAGER7 SPCLIN = GT(FIGSP3,SPCLIN) FIGSP3
  *
  * Terminate the paging activity
  *
  293 PAGER8 NXTFIG(FIGLSTHD) = TEMP1
  294 GE(LINCNT + MINEND - TEMP3 + 1,PAGSIZ)
  + PAGER(LINCNT - TEMP3)
  295 PAGER9 LINSAV = LINCNT :S(RETURN)
  296 END.PAGER

*---------------------------------------------------------------
* Print CNT lines
*---------------------------------------------------------------
  297 DEFINE('PRINTER(TEMP1)', 'PRINTBEG') :S(END.PRINTER)
  * The input output parameters are:
  *  TEMP1 - pointer to the line preceding the first line to be printed
  *  TEMP4 - pointer to a pending overprint line list
  *  TEMP2 - pointer to the first line which has not yet been printed.
  *
  298 PRINTER TEMP2 = NXTLIN(TEMP1)
  299 DIFFER(TEMP2,' ') :F(FRETURN)
  300 TEMP3 = CNT - LINSFC(TEMP2)
  301 TEMP6 = DIFFER(LINFOT(TEMP1),' ') LINFOT(TEMP1) :F(PRINT7)
  *
  * This line refers to a footnote list. See it fits
  *
  302 LINFOT(TEMP1) = ;* erase footnote pointer
  303 TEMP5 = TEMP3 - FIGCNT(TEMP6)
  304 CNT = GE(TEMP5,0) TEMP5 :F(FOT1)
  *
  * This footnote list fits in page
  *
  305 NXTLIN(CURFOT) = FIGBEG(TEMP6)
  306 NXTLIN(FIGLSTHD) = IDENT(FIGLSTHD,CURFOT)
  + LINST(FOTIT1,1,' ', ' ', NXTLIN(FIGLSTHD))
  307 CURFOT = FIGEND(TEMP6) :(PRINT8)
  *
  * The footnote does not fit. If larger than 3 break it.
  *
  308 FOT1 GE(FIGCNT(TEMP6),4) GE(TEMP3,2) :S(FOT2)
  309 LINFOT(TEMP2) = TEMP6 :S(RETURN)
  *
  * The footnote does not fit but may be broken. Compute break
  *
  310 FOT2 TEMP7 = CNT
  311 CNT = LINSFC(TEMP2) ;* at least the line
  312 TEMP7 = GT(TEMP7 - FOTSTT,CNT) TEMP7 - FOTSTT
  313 TEMP7 = TEMP7 - CNT ;* remaining lines
  *
  * Print text to be broken
  *
  314 PRINTER(TEMP1)
A text formatter, version 2

Output a page of lines.
315  TEMP5 = TEMP2                ;* save next line to print
316  TEMP7 = TEMP7 + CNT
317  CNT = 1000
318  OUTPUT = IDENT(FOTLSTHD, CURFCT) FOTIT
319  PRINTER(FOTLSTHD)
320  NXTLIN(FOTLSTHD) =        ;* no more footnotes
     * Print long footnote until break is found
     *
321  CNT = TEMP7
322  FIGBEG(TEMP6) = FIGBEG(TEMP6) - CNT ;* compute remd
323  PRINTER(LINLST("",0,"","",FIGBEG(TEMP6)))
     *
            * TEMP2 points now to the first line in the footnote which has not
            * been printed. Insert it into footnote list
            *
324  FIGBEG(TEMP6) = LINLST(DUPL("",1,PRMRGT(ZIDNFOOT) - 5)
325  + "Cont'd",1,"",TEMP2)
326  FIGCNT(TEMP6) = FIGCNT(TEMP6) + CNT + 1

     * Insert footnote descriptor in list heading
     *
327  TEMP2 = TEMP5
328  LINFOT(PAGHAD) = TEMP6
     * Normal line printing
     *
329  PRINT7 CNT = GE(TEMP3,0) TEMP3 ;F(RETURN)
330  PRINT8 TEMP1 = TEMP2 ;* walk through list
     *
            * Are there any blank lines to give
            *
331  TEMP2 = LINSPEC(TEMP1)
332  GT(TEMP2,1) ;F(PRINT1)
     *
            * Put pending overprints if any
            *
333  DIFFER(TEMP4,"") PENDER(" ") ;F(PRINT3)
334  TEMP2 = TEMP2 - 1
     *
            * Print all space lines
            *
335  PRINT3 TEMP2 = GT(TEMP2,2) TEMP2 - 2 ;F($("PRINT" TEMP2))
336  OUTPUT = '0 ' ;(PRINT3)
     *
            * Print the current line
            *
337  PRINT2 LINSTR(TEMP1) ' ' = '0'
338  PRINT1 OUTPUT = LINSTR(TEMP1)
339  PRINT4 DIFFER(TEMP4,"") PENDER(" ")
     *
            * Collect next line overprints and print currentline overprints
            *
340  PRINTBEG TEMP2 = LINOPB(TEMP1)
341  PRINT5 DIFFER("TEMP2","") ;F(PRINTER)
342  TEMP4 = IDENT(OVRASC(TEMP2),"N") OVRSTR(OVRSTR(TEMP2),
343  + "N",TEMP4) ;S(PRINT6)
344  OUTPUT = OVRSTR(TEMP2)
345  PRINT6 TEMP2 = NXTOVRB(TEMP2) ;(PRINT5)
346  END.PRINTER
*

*----------------------------------------------------------------------------*
* Print pending overprints
* The input output variables are:
* TEMP1 - the control character of the first line to print
* TEMP4 - pointer to the first line to print. On entry TEMP4
* is never NULL, on exit TEMP4 is always NULL.
A text formatter, version 2

Decoding text commands:

* Hexadecimal character insertion pattern definition
  * HEXER = ANY('0123456789ABCDEF') . I1
    + ANY('0123456789ABCDEF') . I2
  * DEFINE('ESCAPEFN(CENCMOD)') : (END.ESCAPEFN)
  * Test if hexadecimal inclusion command
  * ESCAPEFN ESCAPEFN = /* set to null string
    CARD HEXER = : $HEXINC
  * Test if it is an underscore command
    CARD ANY(UDCMD) . I1 = : S(ISUNDER)
  * Test if it is a single overprint command
    CARD ANY(SOVCMD) . I1 = : S(ISSSINOV)
  * Test if word break command
    CARD ANY(SEPCMD) . I1 = : S(SEPAR)
  * Test if word or line fill command
    CARD ANY(FILCMD) . I1 = : S(ISFILL)
  * Test if any cent function command
    CARD ANY(CENCMOD) . I1 = : F(RETURN)
    I1 ANY('PZIUT/') : F(S('CENT' I1))
    ESCAPE = ESCAPE I1 : (FRRETURN)
A text formatter, version 2

Overprint string generating functions.
* Underscore handling function
* Test if underscore was in progression. US CFLG tells that.
  363  TSUNDER GT(US CFLG,0) :F(UND LOOP1)
* Satisfy underscore in progression then reset to no underscore. 
* ESCAPE may contain at most one 'u' and, this, iff underscore is in 
* progression. The memory variables are US CLSTPT, US CFLG and US CPTR. 
  364  US CSTR(US CLSTPT) = DUPL(US CSTR(US CLSTPT), SIZE(WRD) - 
+ US CPF OS(US CLSTPT))
  365  ESCAPE BREAK('u') . I2 'u' = I2 ;* no further 
* Verify if this command initiates a continuous underscore 
  366  UN D LOOP1 US CFLG = 1 ;* set to non continuous
  367  UN D LOOP2 IDENT(I1,'e') :F(UND LOOP3)
  368  US CFLG = 2 ;* set to continuous
  369  CARD ANY(UND CMD) . I1 = :S(UND LOOP2)F(UND LOOP7)
* Test if underscore end command 
* Obtain Underscore descriptor pointer and build an underscore entry 
  370  UN D LOOP3 IDENT(I1,')' ) :F(UND LOOP5)
  371  UN D LOOP7 US CFLG = 0 ;* no more underscore
  372  US CLSTPT = :F(RETURN)
* Processing of end of word underscore alarm 
  373  UN D LOOP5 US CP TR = $('U S C P ' , I1) ;* get pointer 
  374  US CL STHD = US CL ST(ULNSYM(US CP TR),SIZE(WRD),ULNAS C(US CP TR), 
+ US CL ST HD)
  375  US CLSTPT = US CLST HD
  376  ESCAPE = ESCAPE 'u' :F(RETURN)
* There are no further underscores 
  377  ESCAPu US CSTR(US CLSTPT) = DUPL(US CSTR(US CLSTPT),SIZE(WRD) - 
+ US CPF OS(US CLSTPT))
  378  US CL STHD = EQ(US CFLG,2) US CL ST(ULNSYM(US CP TR),0, 
+ US CPLTR,US CL ST HD) :F(UND LOOP6)
  379  US CLSTPT = US CLST HD
  380  E S C TMP = E S C T M P 'u' :(ESCAPEND)
* Single character over print handling 
*  383  ISSN O V SIN PTR = $('ZSI N' , I1)
  384  US CL STHD = US CLST(ULNSYM(SIN PTR),SIZE(WRD) - 1, 
+ ULNASC(SIN PTR),US CLST HD) :F(RETURN)
* Word separating command handling. 
*  385  SEP AR SIN PTR = $( 'ZSE P ' , I1)
  386  SEPLSTHD = US CLST(ULNSYM(SIN PTR),SIZE(WRD),'s',SEPLSTHD)
  387  ESCAPE = ESCAPE 's' :F(RETURN)
* Long over print command handling
*  388  CENT# TEMP1 = 0 ;* Base string flag
  389  TEMP2 = 'c' ;* current line association
A text formatter, version 2

Overprint string generating functions.

390  TEMP3 = SIZE(WRD) ;* where it starts
391  OVFLOOP1 TEMP4 = ;* overprint string
   * Obtain overprint command
   *
392  OVFLOOP2 CARD BREAK('#@&') . I1 LEN(1) . I2 = +
     :S($("OVFN" I2))F(RETURN)
   * The overprint string command is @ or #
   *
393  OVFN#
394  OVFN@  TEMP4 = TEMP4 I1
395  ESCAPEFN = EQ(TEMP1,0) TEMP4        :S($("OVFN1" I2))
396  USCLSTHD = USCLST(TEMP4,TEMP3,TEMP2,USCLSTHD)
397  TEMP2 = 'C'                          :$("OVFN1" I2))
   * Complete case of #
   *
398  OVFN1#                                    :(RETURN)
   * Complete the case for @
   *
399  OVFN1@  TEMP1 = 1
400  CARD '#=' =                              :(OVFLOOP1)
401  TEMP2 = 'N'                              :(OVFLOOP1)
   * Complete the case for @
   *
402  OVFN%  TEMP4 = TEMP4 I1 ESCAPEFN('¥')  :(OVFLOOP2)
A text formatter, version 2

Non delimitting cent commands.

* Hexadecimal inclusion
* HEXINC

403 ESCAPEFN = SUBSTR(SUBSTR('0123456789',112 - CONV(I1) + CONV(I2) + 1,1)
+ (RETURN)

* Double cent command
* CENTF

404 ESCAPEFN = 'F'

* Start a new card without starting a new line
* CENTV

405 CARD =

* Page parameter definition command $G
*

406 CENTG

CARD ('H' . I1 | ANY(123CDNRSZ#')) . I1
+ (SPAN('0123456789') | ' '). I2 ) =
+ $S(S(\$CENTG# I1))F(RETURN)

407 CENTGH

ESCAPE = ESCAPE 'G'

408 CENTGD

PAGSIZ = GT(I2, PAGINI) I2

409 CENTGN

PAGENO = I2

410 CENTGNF

FOTSTT = GT(I2, 4) I2

411 CENTG1

;CENTG2 ;CENTG3
+

412 CENTS

SEQALN = I2

413 CENTS

SPCING = I2

414 CENTS

SPCLIN = I2

415 CENTS

CARD 'P' =

416 CENTS

SPCLIN = SPCLIN + SPCING

417 CENTGC

XYI = I2

418 CENTGC

GT(LINCNT + XYI, PAGSIZ)

419 CENTGC1

(RETURN)

420 CENTGC1

IDENT(MINLIN, 'A') PAGER(LINSAV)

421 CENTGC1

:S(CENTG1)

422 CENTGC1

LE(LINCNT, PAGSIZ) PAGER(LINCNT)

423 CENTGC1

:S(CENTG1)

* String generating functions $J
*

424 CENTJ

CARD ANY('DSPR') . I1 =
+ $S(S(\$CENTJ# I1))F(RETURN)

425 CENTJ

ESCAPE = DATE()

426 CENTJ

ESCAPEFN = RVAL(LINSTR(BUildSTR( )))

427 CENTJ

:CODE(LINSTR(BUildSTR()) ; ;(RETURN)'))>

428 CENTJ

ESCAPE = PAGENC

* Page title reading command $H
*

429 CENTH

CARD ANY('AFMN0123456789') . SAVHAD =
+ :F(RETURN)

430 CENTH

TEMP1 = BUildSTR()

431 CENTH

:* Obtain title string

432 CENTH

LINSTR(TEMP1) ' ' PTab(I) . I1 ' ' = I1

433 CENTH

TITL2 = IDENT(SAVHAD,'A') LINSTR(TEMP1)

434 CENTH

FIFTIT = IDENT(SAVHAD,'F') LINSTR(TEMP1)

435 CENTH

FOTSTR = IDENT(SAVHAD,'M') LINSTR(TEMP1)

436 CENTH

SAVHAD = GT(SAVHAD, PAGINI) PAGINI

* Loop through header lines until title line found
*

437 CENTH

TEMP2 = LINLSTHD

438 CENTH

LINSTR(TEMP1) = EQ(SAVHAD,0) '!' LINSTR(TEMP1) :S(CENTH2)

439 CENTH

LINSTR(TEMP1) = ' ' LINSTR(TEMP1)

440 CENTH

SAVHAD = GT(SAVHAD,0) SAVHAD - 1 :F(CENTH2)

441 CENTH

TEMP2 = NTLIN(TEMP2) :S(CENTH1)

442 CENTH

LINSTR(TEMP2) = LINSTR(TEMP1)
A text formatter, version 2

Non delimitting cent commands.

```
443  LINOVR(TEMP2) = LINOVR(TEMP1)

  + Clear all lower level titles

444  CENTH3 TEMP2 = DIFFER(TEMP2,PAGHAD) NXTLIN(TEMP2)
        :F(RETURN)
445  LINSTR(TEMP2) = ' ' :CEN
446  LINOVR(TEMP2) = :(CENTH3)

  +---------------------------------------------------------------------+
  | Footnote command '%N'. Obtain Footnote mark character or string       |
  +---------------------------------------------------------------------+

447  CEN3T FOTCNT = FOTCNT + 1
448  ESCAPEFN = SUBSTR(FCTSTR,FCTCNT,1)
449  CARD = ESCAPEFN ' ' CARD

  + Collect footnote text

450  FOTIND = FIGFOT(ZIDNFOOT)
451  ESCAPE = ESCAPE 'N' :(RETURN)

  + Collect figure text '%S'. Surround it by delimitting strings

  +---------------------------------------------------------------------+
  | CENTS FIGIND = FIGFOT(ZIDNFIG)
  | NXTLIN(FIGEND(FIGIND)) = LINLST(FIGTIT,FIGS2)
  | FIGEND(FIGIND) = NXTLIN(FIGEND(FIGIND))
  | FIGCNT(FIGIND) = FIGCNT(FIGIND) + FIGSP2

  + Test if figure fits in page

456  TEMP1 = LINCNT + FIGCNT(FIGIND)
457  TEMP2 = TEMP1 ;* verify if previous fig
458  TEMP2 = DIFFER(LINSTR(CURLIN),FIGTIT) TEMP1 + FIGSP1
459  LINCNT = LE(TEMP2,PAGSIZ) IDENT(FIGLSTHD,CURFIG)
        + :S(CENTS1)

  + Figure does not fit in page. Place it in the figure queue

460  NXTFIG(CURFIG) = FIGIND
461  CURFIG = FIGIND
462  FIGIND = :(RETURN)

  + Figure fits in page. Test if preceding is also a figure

463  CENTS1 NXTLIN(CURLIN) = FIGBEG(FIGIND)
464  IDENT(LINSTR(CURLIN),FIGTIT) :S(CENTS2)
465  NXTLIN(CURLIN) = LINLST(FIGTIT,FIGSP1,,NXTLIN(CURLIN))
466  LINCNT = LINCNT + FIGSP1

  + Complete figure insertion

467  CENTS2 CURLIN = FIGEND(FIGIND)
468  FIGIND = ;* delete figure reference
469  SPCLIN = GT(FIGSP3,SPCLIN) FIGSP3 :(RETURN)

  +---------------------------------------------------------------------+
  | Descriptor definition commands '%X'
  |

470  CENTK CARD ANY('IOUBF') . I1 BREAK(')' . I2 = :F(RETURN)
471  CARD (BREAK(')' ) ' )' . I3 = I3 :REM :F(RETURN)
472  LINSTR(BUILDSTR()) (SPAN('**') | **) REM * TEMP1
        + :S($)('CENTK I1))F(RETURN)
473  CENTKI $('ZIND' I2) = EVAL('INDNDEF' TEMP1) :(RETURN)
474  CENTKO $('ZSIN' I2) = EVAL('ULNDEF' TEMP1)
475  SOVCMD = SOVCMD I2 :(RETURN)
476  CENTKU $('ZUSC' I2) = EVAL('ULNDEF' TEMP1)
477  UNDCMD = UNDCMD I2 :(RETURN)
```
A text formatter, version 2

Non delimiting cent commands.

478 CENTKB  $('ZSEP' I2) = EVAL('ULNDEF' TEMP1)  : (RETURN)
479  SEPCMD = SEPCMD I2
480 CENTKF  $('ZFIL' I2) = EVAL('ULNDEF' TEMP1)  : (RETURN)
481  FILCMD = FILCMD I2

*---------------------------------------------------------------------
* Fill word or line to predefined length
* Compute existing length
*
482 ISFILL   /* #JP hook
483 ISFILL0  TEMP1 = DIFFER(I1,'l') DIFFER(I1,'w') 0  :S(ISFILL1)
484 TEMP1 = SIZE(WRD)  /* decrease wordsize
485 TEMP1 = IDENT(I1,'l') TEMP1 + LINLEN  /* decrease line size
*
* Find descriptor name to fill a line
*
486 CARD ANY(FILCMD) . I1 =  :F(RETURN)
*
* Fill with integer multiple of fillstring
*
487 ISFILL1  SINPTR = $('ZFIL' I1)  ;* obtain pointer
488 ESCAPEFN = DUPL(ULNSYM(SINPTR),(ULNASC(SINPTR) - TEMP1)
+  / SIZE(ULNSYM(SINPTR)))
*
* Fill remainder of string with begin portion.
*
489 TEMP1 = ULNASC(SINPTR) - SIZE(ESCAPEFN) - TEMP1 ;* remd len
490 ESCAPEFN = SUBSTR(ULNSYM(SINPTR),SIZE(ULNSYM(SINPTR)) -
+  TEMP1 + 1,TEMP1) ESCAPEFN  : (RETURN)

*---------------------------------------------------------------------
* Table of contents and index generator functions
*
491 CENTY   CARD ANY('IR') . I1 BREAK(' ') . I2 = I2 'T'  :F(RETURN)
492  I2 = LINSTR(BUILDSTR( ))  :$(('CENTY' I1))
493 CENTYI  $(I2) = PAGENO  : (RETURN)
494 CENTYR  ESCAPEFN = $(I2)  : (RETURN)
495 END.ESCAPEFN
A text formatter, version 2

Main program.

* Initialize environment

496 $ANCHOR = 1 ;* always on string start match
497 $STLIMIT = 16000000
498 $EXIT ( $ERRHND ) ;(AROUND)
499 $EXIT ( $ERRHND ) ;* reestablish error exit
500 OUTPUT = ' **ERROR** SERRTYPE 'ST' $LASTNO ' |
+ 
501 CARD SEQN ;(CONTINUE)
502 OUTPUT( ,OUTPUT,,)

* Initialize lists

503 WRDLSTHD = WRDLST()
504 CURWRD = WRDLSTHD
505 FOTLSTHD = LINLST()
506 FIGLSTHD = FIGLST()
507 CURFIG = FIGLSTHD

508 LINLSTHD = LINLST( '1' DUPL( ',51) DATE( ) ' -',0 )
509 PAGHAD = LINLST( '1',1 )
510 NXTLIN(LINLSTHD) = LINLST( '1,1,,LINLST( '1,1,,PAGHAD )
511 PAGINI = 3
512 LINCNT = PAGINI
513 PAGSAY = PAGINI
514 PAGENO = 1
515 CURLIN = PAGHAD
516 TITLE2 = '-'

* Initialize descriptors

* Indentation descriptors

517 ZIDNN = IDNDEF(0,0,68,4,4,4,0,18,1,14,0,5,0,9,0 )
518 ZIDNR = IDNDEF(1,13,68,4,1,15,0,5,1,20,0,5,0,15,0 )
519 ZIDNL = IDNDEF(1,16,68,4,1,15,0,5,1,23,0,5,0,18,0 )
520 ZIDND = IDNDEF(1,19,68,4,1,21,0,5,1,26,0,5,0,21,0 )
521 ZIDNDEF = IDNDEF(2,17,68,4,1,18,0,5,1,23,0,5,0,18,0 )
522 ZIDNFOOT = IDNDEF(1,7,65,0,0,8,0,0,0,13,0,0,0,8,0 )
523 ZIDNFIG = IDNDEF(0,0,68,0,1,6,0,0,0,16,0,0,0,11,0 )

* Underscore descriptors

524 ZUSCn = ULNDEF( '1', 'C' )
525 ZUSCd = ULNDEF( '1', 'C' )
526 ZUSCb = ULNDEF( '1', 'N' )
527 UNDCMD = 'ndb6.'

* Single overprint descriptors

528 ZSINa = ULNDEF( '1', 'C' )
529 ZSING = ULNDEF( '1', 'C' )
530 ZSINf = ULNDEF( '1', 'C' )
531 ZSInt = ULNDEF( '1', 'C' )
532 ZSInu = ULNDEF( '1', 'C' )
533 ZSInC = ULNDEF( '1', 'C' )
534 ZSINs = ULNDEF( '1', 'C' )
535 SOVCMD = 'agiftucbars'

* Word separator descriptors

536 $( 'ZSEP' ) = ULNDEF( '1', 'C' )
537 $( 'ZSEP' ) = ULNDEF( '1', 'C' )
538 SEPCMD = '1,'
A text formatter, version 2

Main program.
* Word or line fill descriptors
*
539     ZFILI = ULNDEF('*',69)
540     FILCMD = 'liw'
*
* Initialize figure and footnote parameters
*
541     FIGTIT = DUPL(' ',14) DUPL(_,50)
542     FIGS1 = 1
543     FIGS2 = 2
544     FIGS3 = 2
545     FOTTIT = DUPL(' ',5) DUPL(_,20)
546     FOSTST = 20
547     FOSTTR = '++,12345678'
*
* Initialize reading environment
*
548     FIND = FINDA
549     CRDEND = ' '  
550     CENCMD = '#GHIJKNPSTUVZY#
551     SEQALN = 90 ;* record no left margin
552     PAGSIZ = 58
553     SPCING = 1 ;* set to double spacing
554     CURIDN = ZIDNN
555     ESCAPE = INDENT() ;* set parameters of line
*
* Read until end of file
*
556     READER()
557     TERMIN('p')
558     LOOP    IDENT(CURLIN,PAGHAD) IDENT(LINFOT(PAGHAD),')') :S(LAAP)
559         PAGER(PAGSIZ) :
560     LAAP    OUTPUT = '1 ===== END OF RUN ====='
561     END
Index

1. Introduction .............................................. 1
2. Basic Concepts .......................................... 2
3. Text Commands ........................................... 6
   3.1 Introduction ....................................... 6
   3.2 Miscellaneous Commands ............................. 8
   3.3 Underscoring and Overprinting .................... 17
   3.4 Figures and Footnotes ............................... 21
   3.5 Unformatted Mode .................................. 26
   3.6 Text Line Affecting Commands .................... 27
   3.7 Page Affecting Commands ........................... 30
4. JCL Needed to Run the Program ...................... 32
5. References ............................................... 32
Appendix 1 Example ....................................... 33
Appendix 2 Summary of Commands ....................... 34
Appendix 3 Hexadecimal Correspondence Tables ....... 36
Appendix 4 System Defined Parameters and Descriptors .. 38