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 $n \geq 0$ 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 MPS should satisfy following relation:

$$MEG \leq MPS$$

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

title, the page number, and a page number follower title. The other header lines will contain just titles [usually blank].

The number of lines per page, i.e. page depth, is controlled by one parameter for all pages [usually 58†]. The page depth counts all lines excluding the first header line.

The number of blank lines to be given preceding a certain kind of line [vertical space parameter], is always increased by the spacing parameter [see section 3.7]. However, spacing commands [ØZ and ØGZ] override the current vertical spacing. The normal value for the spacing parameter is 1, for double spacing. Figures and footnotes are always collected with spacing=0, except, of course, if spacing is redefined [ØGS command, see section 3.7].

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 meaningfull 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:

ERRORn1-STn2 | input record remainder string

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

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 occurence [¢hex, ¢JD, ¢JS, ¢N...¢T, ¢¢, ¢#]. 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, ¢¢, 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 ¢b.

3.2 Miscellaneous commands.

¢¢

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 [hex= two characters of the set {0123456789ABCDEF}] into the word at the place where the command occurs. For example:

a¢AAb¢3Bg¢3C is printed: aαbβgγ

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.

₀	¢80	⁰	¢B0	[	¢AD	]	¢BD	≠	¢BE	α	¢AA	´	¢0F
₁	¢31	¹	¢B1	{	¢8B	}	¢9B	≥	¢AE	β	¢3B	^	¢08
₂	¢32	²	¢B2	⌐	¢AC	¬	¢BC	≤	¢8C	γ	¢3C	`	¢2B
₃	¢53	³	¢B3	¦	¢AB	⌋	¢BB	±	¢9E	þ	¢41	˘	¢0E
₄	¢54	⁴	¢B4	⌐	¢FA	⌐	¢FB	ŧ	¢62	Ł	¢42	~	¢29
₅	¢55	⁵	¢B5	┼	¢8F			ø	¢63	Ø	¢43	˘	¢2C
₆	¢56	⁶	¢B6	¤	¢9C	~	¢EC	đ	¢64	Đ	¢44	¨	¢11
₇	¢57	⁷	¢B7	▪	¢9F	_	¢EB	þ	¢65	Þ	¢45	¸	¢09
₈	¢58	⁸	¢B8	•	¢AF	¨	¢DF	æ	¢66	Æ	¢46	°	¢70
₉	¢59	⁹	¢B9	′	¢51	¯	¢A0	œ	¢67	Œ	¢47	˛	¢EA
(	¢DC	(	¢8D			¿	¢20	ı	¢68	®	¢48	¸	¢CF
)	¢CC	)	¢9D	†	¢73	¡	¢21	¶	¢75	″	¢52	˜	¢EF
+	¢CA	+	¢8E	‡	¢9A	\	¢E0	ȝ	¢D0	Ȝ	¢C0	´	¢69
_	¢BF	−	¢FF					‘	¢CE	’	¢EE	?	¢8A

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

¢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:

abc def ghi klm
nop qrs tuv

¢T{N|¢}

this text command terminates special collection modes. All commands which start a special collection mode, must be followed by this command [¢JS, ¢JP, ¢H, ¢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 occuring in words which do not overflow a line are null commands.

¢KBchar('string')

define or redefine a new word break descriptor.

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.

string is the string which is to be appended to the first portion of the broken word. string may be the null string.

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

char	string	defining command
-	-	¢KB-('-')
,		¢KB,('')

Figure 3.2 Table of system defined word break descriptors.

The following lines of input:

```
¢KITS(0,0,40,0,0,30,0,0,0,30,0,0,0,30,0) ¢ITS
¢KB%('%%')
ab¢-cd ef¢,gh ij kl¢-mn¢-opqr st¢,uvwx
¢bxy¢%zabcdefg
```

produce following output:

```
abcd  efgh
ij   klmn-
opqr    st
uvwx  xy%%
zabcdefg
```

¢JD

this text command inserts the date into the word at the place of occurence. 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*string*¢T

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

abc¢JS DUPL('*',5) ¢Tefg is printed:

abc*****efg

¢JP*string*¢T

in this command *string* must be a valid SNOBOL4 program section. When ending, this program should either flow through the end, or use the label 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 *string* has been collected, the program represented by *string* is compiled and executed [see CODE function in SNOBOL4]. For example the string

abc¢JP ESCAPEFN = 'hm hm hm hm' ¢Tdef

generates the next strange output

abchm hm hm hmdef

¢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

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 MEG<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.

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

¢YRstring¢

insert the value associated with string into the text. This text command retrieves the value associated with string by a ¢YI text command.

Example:
abc¢YIxx def¢YRxx ghi is printed: abcdef14ghi

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.

¢KFchar('string',size)

define or redefine a word or line fill descriptor.

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.

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

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.

char	string	size	defining command
i	.	69	¢KFi('.',69)

Figure 3.3 Table of the system defined word or line fill descriptor.

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:

¢KFz('¢70',1)

defines a new line and word fill descriptor. This descriptor will reserve one single space, and print the character • in this space. It is then quite easy to spot

and introduce the required character by means of a typewriter.

Examples:

¢Z1 ab cd ef¢lig ij is printed:

ab cd

ef.....g ij

¢Z1 ab cd ef¢wig ij is printed:

ab cd

ef.....g

ij

Definition of 'z' ¢KFz('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 succeede such 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 sigle 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.

¢KUchar('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.

char	string	assoc	defining command
n	_	C	¢KUn('_','C')
d	_	C	¢KUd('¢EB','C')
b	~	N	¢KUb('¢EC','N')

Figure 3.4 Table of the system defined underscore descriptors.

Some examples:

ab ¢nword xyz is printed: ab word xyz

ab ¢&nde fg¢. hi is printed: ab de fg hi

¢KU*('*','N')

isn¢&n't th¢&*at nice¢.? is printed: isn't that nice?
** ****

¢char ¢{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.

¢KOchar('string','assoc')

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.

char	string	assoc	defining command
a	´	C	¢KOa('¢0F','C')
g	`	C	¢KOg('¢2B','C')
f	^	C	¢KOf('¢08','C')
t	~	C	¢KOt('¢29','C')
u	¨	C	¢KOu('¢11','C')
c	¸	C	¢KOc('¢09','C')
s	/	C	¢KOs('/','C')

Figure 3.5 Table of the system defined constant overprint descriptors.

Examples:

fac¢cade is printed: façade

¢KO|('|','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:

string1||base-string||string2 †

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 overi 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@///@#|||#ghi is printed: abcdefghi
|||

The string: ¢#user defined separation !!!#
is printed: user defined separation !!!

3.4 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. ¢Tnote

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 refering 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 refering to it is on the same text page. Thus it is possible to have figures refering to footnotes, without that the synchronism is violated.

¢Stext¢T

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 ¢IFIG 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.

¢Ntext¢T

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.

¢HF*string*¢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.

¢G1*nn*

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].

¢G2*nn*

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

¢G3*nn*

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].

¢HN*string*¢T

this command sets the footnote delimiting line to *string*. Its normal value can be seen in the footnote reference containing pages in this text.

¢HM*string*¢T

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 acces 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].

¢GN*nn*

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 ½ 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.

¢KInam(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:

¢KIN(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}nn

This command terminates the current line and, then, starts a new line of type continuation line which will be preceded by nn blank lines. If the value of nn 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

parameter	indentation descriptor						
	N	R	L	D	DEF	FOOT	FIG
pw	0	1	1	1	2	1	0
ir	0	13	16	19	17	7	0
rm	68	68	68	68	68	65	68
me	4	4	4	4	4	0	0
vsf	4	1	1	1	1	0	0
lmf	4	15	18	21	18	8	6
ssf	0	0	0	0	0	0	0
tbf	18	5	5	5	5	0	0
vsp	1	1	1	1	1	0	0
lmp	14	20	23	26	23	13	16
ssp	0	0	0	0	0	0	0
tbp	5	5	5	5	5	0	0
vsc	0	0	0	0	0	0	0
lmc	9	15	18	21	18	8	11
ssc	0	0	0	0	0	0	0

```
¢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)
¢KIDEF(2,17,68,4,1,18,0,5,1,23,0,5,0,18,0)
¢KIFOOT(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, nn+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 skiped 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 ¢ZI0 ¢ZInn-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.

¢Hn string¢T

This command sets the nth header line to string, i.e. it sets titles and subtitles. The subtitles in the page header lines i>n are set to blank. The first line in the page header is n=0. Overprints underscoring are all valid within 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 SPITBCL,REGION.GO=200K,PARM.GO='R=15K'
//GO.SYSIN DD DSN=file-name
//         DD *
```

in 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.

[1] Griswold, R.E.; Poage, J.F.; Polonsky, I.P. The SNOBOL4 Programming Language; Prentice Hall Inc.; second edition; 1971

[2] Dewar, R.B.K. SPITBOL; Illinois Institute of Technology; Feb 1971

[3] von Staa, A. AOTF - Arndt's Own Text Formatter; Sep 1973; Unpublished

Appendix 1.

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

```
¢H0 ¢T ¢G# ¢HA ¢T  ¢ZI0
¢ZI18 ¢KU*('*','N')
¢JSDUPL(' ',20)¢T¢*¢#A TEXT FORMATTER#
¢ZI10
¢JSDUPL(' ',33)¢Tby Arndt von Staa
¢Z0¢JSDUPL(' ',36)¢TUniversity of Waterloo
¢Z0¢JSDUPL(' ',36)¢T June 1974
¢GH ¢G#1
¢H0¢JSDUPL(' ',28)¢TA Text Formatter                    -¢T
¢HA-¢T  ¢IN
1. ¢nIntroduction¢..  ¢YI1.
 This text formatter is a second generation text formatter. It
improves on the first generation formatter¢AD3¢BD 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:
¢IR i- capabilities to collect figures and footnotes and placing them
in the page text;
¢IR ii- allow overprinting, underscoring and accentuation mark
insertion
by means of easy to use commands;
¢IR 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 th
line length is exceeded. The right margin is usually aligned;
¢IR iv- to allow both word shifting as in (iii) above, or  to
generate lines which are pseudo input record images;
¢IR v- to be completely parametric, i.e. any formatting parameter may
be defined or redefined at execution time.
¢IN ¢P 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 mnemonicity
of the text commands. Still, the author believes it to be
a worthwhile investment to learn to use this text formatter.
¢IN 2. ¢&nBasic Concepts¢..  ¢YI2.
 In this section we will define the terminology used throughout this
text.
 The input text consists of ¢nwords and ¢&ntext commands¢.. A text
command is one of several well defined strings preceded by the
¢&nescape 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, ¢¢I, ¢¢P,
¢¢T, ¢¢U, ¢¢Z and ¢¢/¢BD.
 The 8 last characters of an input record will ¢dalways 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 ¢nmodes¢.:
¢nformatted and ¢nunformatted¢.. In the formatted 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

Command	Page	Meaning
¢00 ... ¢FF	-8-	hexadecimal character inclusion.
¢GCn	-30-	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.
¢GDn	-31-	set page depth to n.
¢GHn	-30-	terminate the current line. Skip to a new page if the current page is not empty. If 0≤n≤9, set the number of header lines to n.
¢GNn	-25-	set the maximal footnote size per page to n.
¢GRn	-9-	set the record number left margin to n.
¢GSn	-31-	set the line spacing to n. [0 is single spaced, 1 is double spaced].
¢GZn	-30-	set the spacing preceding the current line to n. Do not terminate the current line.
¢G#n	-31-	set the page number of the current page to n.
¢G1n	-24-	set the spacing preceding the figure delimiter line to n.
¢G2n	-24-	set the spacing after the figure text and preceding the figure delimiter line to n.
¢G3n	-24-	set the spacing after a figure delimiter and the text succeeding the figure to n.
¢HA*string*¢T	-31-	set the page number follower title to *string*.
¢HF*string*¢T	-24-	set the figure delimiter string to *string*.
¢HM*string*¢T	-25-	set the footnote marker string to *string*.
¢HN*string*¢T	-25-	set the footnote delimiter string to *string*.
¢Hn*string*¢T	-32-	set the title of the header line 0≤n≤9 to *string*. Set all titles m'>n to null.
¢I*nam*	-27-	terminate current line and text block. Turn *nam* into the current indentation descriptor.
¢JD	-12-	insert the SPITBOL date into the text.
¢JP*string*¢T	-13-	obtain and execute the SPITBOL program section *string*.
¢JS*string*¢T	-13-	obtain, evaluate and place into the text of the current word, the SPITBOL object *string*.
¢J#	-13-	insert the current page number into the text.
¢KB*char*(...)	-12-	definition of the word break descriptor *char*.
¢KF*char*(...)	-16-	definition of the word or line fill descriptor *char*.
¢KI*nam*(...)	-27-	definition of the indentation descriptor *nam*.
¢KO*char*(...)	-19-	definition of the constant text overprint descriptor *char*.
¢KU*char*(...)	-18-	definition of the underscore descriptor *char*.
¢N*text*¢T	-23-	generate a footnote with text *text*.
¢P	-27-	terminate the current line and text block. Initiate a new paragraph.
¢S*text*¢T	-23-	generate a figure with text *text*.
¢T	-11-	special collection mode termination command.
¢U*text*¢T	-26-	collect *text* in unformatted mode.
¢V	-10-	terminate the current input record, proceed from the next input record without signalizing end of record.
¢YI*string*¢	-14-	associate the current page number with *string*.
¢YR*string*¢	-14-	insert the value associated with *string* into the text.

¢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.
¢#<u>string</u># -20- collect <u>string</u> in unformatted mode, and generate overprints for substrings initiated with @.
¢¢ -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.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				•		þ			^	˛					˘	ˊ
1		¨														
2	¿	ı			ð	þ				˜		ˋ	˘			
3		1	2									β	γ			
4		þ	Ł	Ø	Đ	Þ	Æ	Œ	®	•	¢	.	<	(	+	|
5	&	′	″	3	4	5	6	7	8	9	!	$	*	)	;	¬
6	-	/	ł	ø	d	þ	æ	œ	ı	˝	£	,	%	_	>	?
7	°			†		‟	?				:	#	@	'	=	"
8	o	a	b	c	d	e	f	g	h	i	?	{	≤	‹	+	┼
9		j	k	l	m	n	o	p	q	r	‡	}	¤	›	±	■
A	¯		s	t	u	v	w	x	y	z	α	˪	˩	[	≥	•
B	0	1	2	3	4	5	6	7	8	9		˫	˥	]	≠	_
C	3	A	B	C	D	E	F	G	H	I	+		)		‘	‚
D	3	J	K	L	M	N	O	P	Q	R			(		•	„
E	\		S	T	U	V	W	X	Y	Z	.	–	~	•	’	‾
F	0	1	2	3	4	5	6	7	8	9	˗	ˉ				–

0 ¢80	0 ¢B0	[¢AD	] ¢BD	≠ ¢BE	α ¢AA	´ ¢0F
1 ¢31	1 ¢B1	{ ¢8B	} ¢9B	≥ ¢AE	β ¢3B	^ ¢08
2 ¢32	2 ¢B2	⌐ ¢AC	¬ ¢BC	≤ ¢8C	γ ¢3C	` ¢2B
3 ¢53	3 ¢B3	½ ¢AB	¼ ¢BB	± ¢9E	ƀ ¢41	˘ ¢0E
4 ¢54	4 ¢B4	« ¢FA	» ¢FB	ŧ ¢62	Ł ¢42	~ ¢29
5 ¢55	5 ¢B5	┼ ¢8F		ø ¢63	Ø ¢43	˘ ¢2C
6 ¢56	6 ¢B6	¤ ¢9C	¨ ¢EC	đ ¢64	Đ ¢44	¨ ¢11
7 ¢57	7 ¢B7	■ ¢9F	¯ ¢EB	þ ¢65	Þ ¢45	¸ ¢09
8 ¢58	8 ¢B8	• ¢AF	¸ ¢DF	æ ¢66	Æ ¢46	° ¢70
9 ¢59	9 ¢B9	′ ¢51	¯ ¢A0	œ ¢67	Œ ¢47	˛ ¢EA
(¢DC	(¢8D		¿ ¢20	ı ¢68	ø ¢48	¸ ¢CF
) ¢CC	) ¢9D	† ¢73	¡ ¢21	ł ¢75	″ ¢52	¯ ¢EF
+ ¢CA	+ ¢8E	‡ ¢9A	\ ¢E0	ȝ ¢D0	Ȝ ¢C0	˝ ¢69
_ ¢BF	− ¢FF			‘ ¢CE	’ ¢EE	’ ¢8A

Appendix 4.

System defined parameter values

No. of header lines - 3 (¢GH3)
Page number - 1 (¢G#1)
Page title - <51 blanks> <date> <10 blanks> -
Page no. follower title - - (¢HA-¢T)
Record no. left margin - 90 (¢GR90)
Figure spacings - ¢G11 ¢G21 ¢G32
Footnote size - 20 (¢GN20)
Footnote markers - ¢HM†‡*[12345]¢T
Page depth - 58 (¢GD58)
Line spacing - 1 (¢GS1)
Current indentation - N (¢IN)

System defined descriptors

char	string	defining command
-	-	¢KB-('-')
,		¢KB,('')

Table of the system defined word break descriptors.

char	string	size	defining command
i	.	69	¢KFi('.',69)

Table of the system defined word or line fill descriptor.

char	string	assoc	defining command
n	_	C	¢KUn('_','C')
d	–	C	¢KUd('¢EB','C')
b	~	N	¢KUb('¢EC','N')

Table of the system defined underscore descriptors.

char	string	assoc	defining command
a	´	C	¢KOa('¢0F','C')
g	`	C	¢KOg('¢2B','C')
f	^	C	¢KOf('¢08','C')
t	~	C	¢KOt('¢29','C')
u	¨	C	¢KOu('¢11','C')
c	¸	C	¢KOc('¢09','C')
s	/	C	¢KOs('/','C')

Table of the system defined constant overprint descriptors.

parameter	indentation descriptor						
	N	R	L	D	DEF	FOOT	FIG
pw	0	1	1	1	2	1	0
ir	0	13	16	19	17	7	0
rm	68	68	68	68	68	65	68
me	4	4	4	4	4	0	0
vsf	4	1	1	1	1	0	0
lmf	4	15	18	21	18	8	6
ssf	0	0	0	0	0	0	0
tbf	18	5	5	5	5	0	0
vsp	1	1	1	1	1	0	0
lmp	14	20	23	26	23	13	16
ssp	0	0	0	0	0	0	0
tbp	5	5	5	5	5	0	0
vsc	0	0	0	0	0	0	0
lmc	9	15	18	21	18	8	11
ssc	0	0	0	0	0	0	0

```
¢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)
¢KIDEF(2,17,68,4,1,18,0,5,1,23,0,5,0,18,0)
¢KIFOOT(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)
```

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:
      *       WRDSTR - 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.
      *       USCFLG - 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(LINSTR,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.
```

```
A text formatter, version 2

Definition of list nodes.
      *
      *       PAGHAD - pointer to the last line of the page header line list.
      *                It is modified only by '¢GH' commands.
      *       PAGINI - counter of the physical lines in the page header list.
      *                It is modified only by '¢GH' 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(OVRSTR,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('FIGLST(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:
      *      FIGLSTHD - pointer to the figure list header node. Remains
      *                 unchanged.
      *      CURFIG - pointer to the last figure entry. Must be FIGLSTHD if no
      *               figure exists.
      *      LINFOT( ) - associates next line with footnote list.
      *      FIGIND - pointer to the figure descriptor just built
      *      FOTIND - pointer to the footnote list descriptor just built
      *      FIGFOT( ) - function returning a figure or footnote descriptor
      *
      *-------------------------------------------------------------------
      * Underscore and sigle overprint descriptor
      *
    6       DATA('ULNDEF(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 sigle overprint descriptor 'char'.
      *
      *-------------------------------------------------------------------
      * Indentation block entry definition
      *
    7       DATA('IDNDEF(INTCNT,PRMRGT,RGTMRG,BLKEND,SPACF,LEFTF,MINLF,TXTMF,'
      +            'SPACP,LEFTP,MINLP,TXTMP,SPACC,LEFTC,MINLC)')
      *
      * 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.
       *      RGTMRG - 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 sigle spaced left string length of
       *                each of the three line types.
       *      TXTM (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(' ¢') . FOUNDWRD
     9      FINDB = BREAK('¢') . 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.
      *      WRDCNT - 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 RTAB(8) . 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 delimitted by an escape
      *
    15 FINDER     CARD FIND =                           :F(READER)
    16            WRD = WRD FOUNDWRD                    ;* FOUNDWRD is generated by FIND
    17            CARD '¢' =                            :F(PUTWRD)
    18            WRD = WRD ESCAPEFN(CENCMD)                    :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 = USCPOS(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) -
     +                  USCPOS(USCLSTPT))
      *
      * Break the word into two portions
      *
    28 LINR2      LINLEN = TEMP2 + 1                    ;* there is a space after
    29            WRDCNT = WRDCNT + 1                   ;* account word
    30            NXTWRD(CURWRD) = WRDLST('')
    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 oveprints 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       WRDOVR(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 = WRDCNT + 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(ESCND)
   54 ESCAPRET    WRD =                            :(FINDER)
```

A text formatter, version 2

Escape handling within READER( )

```
      *
      * The used variables are:
      *     ESCAPE - a lsit of single character escape commands.
      *     ESCTMP - a list of single character escape commands which is
      *              to be followed after the next word is completed.
      *
      * Distribute the escape commands
      *
   55 ESCND       ESCTMP =                          ;* initialize result string
   56 ESCAPEND    ESCAPE LEN(1) . I1 =              :S($('ESCAP' I1))
   57             ESCAPE = ESCTMP                   :(ESCAPRET)
      *
      *-------------------------------------------------------------------
      *ESCAPu - see underscore command handling in overprint functions.
      *
      *-------------------------------------------------------------------
      * Primal word handling.
      * Initialize parameters. TEMP1 ¬= NULL if WRD¬=''
      *
   58 ESCAPp      TEMP1 = NXTWRD(WRDLSTHD)
   59             DIFFER(WRD,'')                    :F(PRIM4)
   60             DIFFER(TEMP1,CURWRD)              :F(PRIM3)
      *
      * Combine 2 words into one single word
      *
   61             TEMP2 = SIZE(WRDSTR(TEMP1)) + 1
   62             WRDSTR(TEMP1) = WRDSTR(TEMP1) ' ' WRDSTR(CURWRD)
      *
      * Correct USCPOS entries of all overprint list entries of 2nd word
      *
   63             TEMP3 = WRDOVR(CURWRD)
   64             DIFFER(TEMP3,'')                  :F(PRIM2)
   65 PRIM1       USCPOS(TEMP3) = USCPOS(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(CURWRD)
      *
      * Reset word list pointers to one only word so far
      *
   69 PRIM2       CURWRD = TEMP1
   70             NXTWRD(TEMP1) =
      *
      * Count primal words and test if end
      *
   71 PRIM3       PRMCNT = PRMCNT - 1
   72 PRIM4       ESCTMP = GT(PRMCNT,0) ESCTMP 'p'       :S(ESCAPEND)
   73             LINSIZ = RGTMRG(CURIDN)
   74             WRDCNT = DIFFER(TEMP1,'') 1            :F(ESCAPEND)
   75             LINLEN = LINLEN - SIZE(WRDSTR(CURWRD)) - 1      :(ESCAPEND)
      *
      *-------------------------------------------------------------------
      * Paragraph escape handling 'ØP'
      *
   76 ESCAPP      TERMIN('P')                       :(ESCAPEND)
      *
      *-------------------------------------------------------------------
      * Indentation command escape handling 'ØInam(bk|Ø)'
      * Complete line and finish previous block if nescessary
      *
   77 ESCAPI      TERMIN('F')                       ;* complete line
      *
      * obtain indentation descriptor
      *
   78             CARD BREAK(' Ø') . I1 = DIFFER($('ZIDN' I1),'') :F(ESCAPEND)
   79             CURIDN = $('ZIDN' I1)             ;* 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     LINPARSET('C')
   86             CARD ('I' | '') . I1 ('R' | '') . I2
      +                 SPAN('0123456789') . SPCLIN =          :F(ESCAPEND)
   87             SPCLIN = DIFFER(I2,'') SPCLIN + SPCING
   88             DIFFER(I1,'')                   :F(ESCAPEND)
      *
      * Construct inconditional space
      *
   89             NXTWRD(WRDLSTHD) = WRDLST('  ')
   90             LINER(1000)                     :(ESCAPEND)
      *
      *-------------------------------------------------------------------
      * Definition of a new set of page header lines '¢GH(n)'
      *
   91 ESCAPG      TERMIN('P')
   92             LINPARSET('C')
   93             DIFFER(CURLIN,PAGHAD) 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             PAGSAV = PAGINI
   98             PAGHAD = LINLSTHD
   99             TEMP1 =
  100             PAGHAD = GT(PAGINI,0) LINLST('  ',1,'','','')   :F(ESCAPG2)
      *
      * Create PAGINI-2 lines
      *
  101             TEMP1 = PAGHAD
  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 = PAGHAD                 :(ESCAPEND)
      *
      *-------------------------------------------------------------------
      * Termination command escape handling '¢T(N|bk)'
      *
  106 ESCAPT      CARD ('N' *TERMIN('P') | ' ') =
  107             LINER(1000)                     ;* done if not end of block
  108             LINPARSET('C')                  ;* set to continuation line
  109             FNSTACK LEN(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             LINPARSET('C')                  ;* by defn continuation line
      *
      * Set unformatted mode parameters
      *
  113             FIND = FINDB                    ;* no blanks as delimitters
  114             CRDEND = '¢/'                   ;* force end of word
```

Escape handling within READER( )

```
  115             WRD =                           ;* erase contents of word
  116             FNSTACK = 'U' FNSTACK           ;* ¢T may proceede
  117             READER( )                       ;* read from new card
      *
      * Reset formatted mode parameters. This point is reached by '¢T' return
      *
  118             FIND = FINDA                    ;* blanks are delimitters 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')
  123             CARD =                          :(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 preceede 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(FRETURN)
  131             LIN = ' '  DUPL(' ',LFTMRG)      ;* Beginning of line is blank
      *
      * Put the intermediate right margin in if there is any.
      *
  132             GT(INTRGT,0)                   :F(INLEFT)
  133                   LIN = ' '  DUPL(' ',INTRGT - SIZE(WRDSTR(CURWRD)))
      +                       WRDSTR(CURWRD)
  134                   DIFFER(WRDOVR(CURWRD)) OVERLAYER(CURWRD)
  135                   LIN = LIN DUPL(' ',LFTMRG - INTRGT)
  136                   CURWRD = NXTWRD(CURWRD)
  137                   WRDCNT = WRDCNT - 1    ;* one word is in already
  138                   INTRGT = 0             ;* inhibit further primals
  139                   ESCAPE BREAK('p') . I1 'p' = I1
      *
      * Put the initial single spaced section of the line in
      *
  140 INLEFT      DIFFER(CURWRD,'')              :F(PUTLIN)
  141             LIN = LIN WRDSTR(CURWRD)
  142             DIFFER(WRDOVR(CURWRD),'') OVERLAYER(CURWRD)
  143             CURWRD = NXTWRD(CURWRD)
  144             WRDCNT = WRDCNT - 1
      *
      * Test if initial portion of line has been exausted
      *
  145             LIN = LT(SIZE(LIN),LFTMIN) LIN ' '     :S(INLEFT)
  146             DIFFER(CURWRD,'')              :F(PUTLIN)
      *
      * Compute space factor for weighted extra blank insertion
      *
  147             SPFCT = (100 * (LINSIZ - LINLEN + 1)) / WRDCNT + 100
  148             OLDSPC = 0
  149             WRDCNT = 1
      *
      * Put word in line preceding it by sufficient spaces
      *
  150 ALLWRDS     NEWSPC = (WRDCNT * SPFCT + 52) / 100
  151             LIN = LIN DUPL(' ',NEWSPC - OLDSPC) WRDSTR(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,'') WRDCNT + 1        :S(ALLWRDS)
      *
      * Line building has been completed. Put line into page list
      *
  156 PUTLIN      DIFFER(MINLIN,'') DIFFER(MINLIN,'A')              :F(PUTL0)
      *
      * It is the first line of a text block. Test if block fits
      *
  157 PUTL3       GT(LINCNT + MINLIN,PAGSIZ)  :F(PUTL2)
  158             LE(LINCNT,PAGSIZ) PAGER(LINCNT)          :S(PUTL3)
  159             PAGER(LINCNT - MINEND)                   :(PUTL3)
  160 PUTL2       MINLIN =                                 :(PUTL0)
      *
      * Introduce the record number into the line
      *
  161 PUTL0       LIN = DIFFER(CRDEND,'ȼV') LIN DUPL(' ',SEQALN -
      +                   SIZE(LIN)) SEQN
      *
      * Account this line in the line list
      *
  162 PUTL1       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
      *
  177 LINPARSET  INTRGT = 0
  178            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('MINL' TMP '(CURIDN)') + LFTMRG
  183            LINLEN = LFTMRG
  184            IDENT(TMP,'C')                 :S(RETURN)
      *
      * Set block parameters
      *
  185            MINEND = BLKEND(CURIDN)
  186            MINLIN = EVAL('TXTM' 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('BUILDSTR()INTRGT,PRMCNT,LINSIZ,MINEND,SPCLIN,LFTMRG,'
     +           'LFTMIN,MINLIN,FIND,ESCAPE,CRDSAV,WRDLSTHD,CURWRD,WRDCNT,'
     +           'LINLEN,WRD,USCLSTHD,USCLSTPT,USCFLG,USCPTR,LINLSTHD,'
     +           'CURLIN,LINCNT,PAGSIZ,LINSAV,FOTIND,SPCING,CENCMD,I1,I2')
     +                                         :(END.BUILDSTR)
      *
      * Inhibit line building during string building
      *
  194 BUILDSTR   LINLEN = 0                    ;* set fake line parameters
  195            LFTMRG = 0
  196            LFTMIN = 1000                 ;* guarantee no line building
  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)) . I1 CRDEND = I1 '¢V'
  203            CRDEND = '¢V'
      *
      * 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             FINDER( )                       ;* 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             BUILDSTR = CURLIN               :(RETURN)
  214 END.BUILDSTR
      *
      *-----------------------------------------------------------------------
      * Terminate a block with a paragraph command
      *
  215       DEFINE('TERMIN(TEMP1)')               :(END.TERMIN)
  216 TERMIN     LINER(1000)                       ;* force end of line
      *
      * Test if complete block goes on one page
      *
  217             IDENT(MINLIN,'A') GT(LINCNT,PAGSIZ) PAGER(LINSAV)
      *
      * Redefine line and page parameters
      *
  218             LINSAV = LINCNT                 ;* forsee case no paging
  219             DIFFER(TEMP1,'F') LINPARSET('P')      :(RETURN)
  220 END.TERMIN
      *
      *-----------------------------------------------------------------------
      * Set all parameters for a new indentation block
      *
  221       DEFINE('INDENT( )')                     :(END.INDENT)
  222 INDENT     LINPARSET('F')                    ;* block and line parameters
      *
      * Set constant indentation block parameters
      *
  223             INTRGT = PRMRGT(CURIDN)         ;* primal right margin
  224             PRMCNT = INTCNT(CURIDN)         ;* no. of primal words
  225             PRMCNT = EQ(INTRGT,0) 0         ;* prevent definition errors
  226             LINSIZ = RGTMRG(CURIDN)
      *
      * Prepare primal words parameters
      *
  227             INDENT = GT(PRMCNT,0) 'p'       :F(RETURN)
  228             LINSIZ = 1000                   :(RETURN)
  229 END.INDENT
      *
      *-----------------------------------------------------------------------
      * Collect footnote and figure text. This function serves only to save
      *and initialize values
      *
  230       DEFINE('FIGFOT(CURIDN)INTRGT,PRMCNT,LINSIZ,MINEND,SPCLIN,LFTMRG,'
      +             'LFTMIN,MINLIN,FIND,CRDEND,ESCAPE,WRDLSTHD,CURWRD,WRDCNT,'
      +             'LINLEN,WRD,USCLSTHD,USCLSTPT,USCFLG,USCPTR,LINLSTHD,'
      +             'CURLIN,LINCNT,PAGSIZ,LINSAV,FOTIND,SPCING,CRDSAV')
  231                                             :(END.FIGFOT)
  232 FIGFOT      PAGSIZ = 1000                   ;* inhibit paging
  233             WRDLSTHD = WRDLST( )
  234             CURWRD = WRDLSTHD
  235             LINLSTHD = LINLST( )
  236             CURLIN = LINLSTHD
  237             ESCAPE = INDENT( )              ;* set parameters of line
      *
      * Set remaining parameters
      *
  238             FIND = FINDA                    ;* blanks delimmit
  239             CRDEND = ' '                    ;* normal paragraph
  240             FNSTACK = 'N' FNSTACK           ;* allow '¢T' to operate
  241             FINDER( )
```

Auxiliary functions.

```
      *
      * The figure and footnote have been completed reset
      *
  242             FIGFOT = FIGLST(NXTLIN(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 currntly being formed by LINER( )
      *
  244      DEFINE('OVERLAYER(WRD)')              :(END.OVERLAYER)
      *
      * Initialize
      *
  245 OVERLAYER  POS = SIZE(LIN) - SIZE(WRDSTR(WRD))
  246            WRD = WRDOVR(WRD)               ;* enter overprint string start
      *
      * Loop through all overprint strings of the word
      *
  247 OVRLOOP2   DIFFER(WRD,'')                  :F(RETURN)
  248            TMP = POS + USCPCS(WRD)
      *
      * Loop through overprint lines in search for one where string fits
      *
  249            PTR = OVRLSTHD
  250 OVRLOOP1   DIFFER(PTR,'')                  :F(OVRLOOP5)
      *
      * This line is valid iff line associations are equal and field is blank
      *
  251            OVRSTR(PTR) IDENT(OVRASC(PTR),USCASC(WRD)) LEN(TMP) . I1
      +                 DUPL(' ',SIZE(USCSTR(WRD)))
      +                 = I1 USCSTR(WRD)         :S(OVRLOOP4)
      *
      * get next line. If lines are exausted, build a new line.
      *
  252            PTR = NXTOVR(PTR)               :(OVRLOOP1)
  253 OVRLOOP5   OVRLSTHD = OVRLST(('+' DUPL(' ',TMP - 1) USCSTR(WRD)
      +                 DUPL(' ',120 - TMP - SIZE(USCSTR(WRD)))),
      +                 USCASC(WRD),OVRLSTHD)
      *
      * Get next overprint string in word
      *
  254 OVRLOOP4   WRD = NXTUSC(WRD)               :(OVRLOOP2)
  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
      *     PAGHAD - 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(LINLSTHD) PAGENO TITLE2
  258             PAGENO = DIFFER(PAGENO,'') PAGENO + 1
      *
      * Get ready to print CNT lines
      *
  259             NXTLIN(FOTLSTHD) =              ;* no footnote on this page
  260             CURFOT = FOTLSTHD
  261             TEMP4 =
  262             LINCNT = LINCNT - CNT
  263             PRINTER(LINLSTHD)               :S(PAGER1)
      *
      * The list has been exausted.
      *
  264             CURLIN = PAGHAD                 ;* page heading only
  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(PAGHAD) = TEMP2          ;* delete printed lines
      *
      * Prepare to print all the footnote lines
      *
  269             CNT = 1000
  270             PRINTER(FOTLSTHD)               ;* print collected footnotes
  271             NXTLIN(FOTLSTHD) =
  272             FOTCNT = 0
  273             DIFFER(TEMP4,'') PENDER(' ')
      *
      * Initialize pointers to figure and figure insertion
      *
  274             TEMP1 = DIFFER(NXTFIG(FIGLSTHD)) NXTFIG(FIGLSTHD)
      +                                           :F(PAGER9)
  275             TEMP3 = LINCNT - PAGINI         ;* figure goes on top
  276             TEMP4 = PAGHAD                  ;* top of page
  277             NXTFIG(FIGLSTHD) =              ;* avoid recursive figures
  278             FIGBEG(TEMP1) = LINLST(FIGTIT,0,,,FIGBEG(TEMP1))
      *
      * Insert the figure into the page. TEMP4 points to last line
      *
  279 PAGER4      NXTLIN(FIGEND(TEMP1)) = NXTLIN(TEMP4) ;* put continuation
  280             NXTLIN(TEMP4) = FIGBEG(TEMP1)         ;* figure is in now
  281             CURLIN = IDENT(CURLIN,TEMP4) FIGEND(TEMP1)
  282             TEMP4 = FIGEND(TEMP1)                 ;* last line = TEMP4
  283             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(TEMP1)
  286             CURFIG = IDENT(TEMP1) FIGLSTHD          :S(PAGER6)
  287             LE(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 = LINSPC(NXTLIN(TEMP4))
  290             LINCNT = LT(TEMP2,FIGSP3) LINCNT - TEMP2 + FIGSP3
      +                                         :F(PAGER8)
  291             LINSPC(NXTLIN(TEMP4)) = FIGSP3          :(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                 :(RETURN)
  296 END.PAGER
      *
      *-----------------------------------------------------------------------
      * Print CNT lines
      *
  297       DEFINE('PRINTER(TEMP1)','PRINTBEG')          :(END.PRINTER)
      * The input ouput 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 - LINSPC(TEMP2)
  301             TEMP6 = DIFFER(LINFOT(TEMP1),'') LINFOT(TEMP1)  :F(PRINT7)
      *
      * This line refers to a footnote list. See if 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(FOTLSTHD) = IDENT(FOTLSTHD,CURFOT)
      +                 LINLST(FOTTIT,1,'','',NXTLIN(FOTLSTHD))
  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           :(RETURN)
      *
      * The footnote does not fit but may be broken. Compute break
      *
  310 FOT2        TEMP7 = CNT
  311             CNT = LINSPC(TEMP2)             ;* at least the line
  312             CNT = 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,CURFOT) FOTTIT
  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(' ',PRMRGT(ZIDNFOOT) - 5)
     +                "Cont'd",1,'','',TEMP2)
  325             FIGCNT(TEMP6) = FIGCNT(TEMP6) + CNT + 1
      *
      * Insert footnode descriptor in list heading
      *
  326             TEMP2 = TEMP5
  327             LINFOT(PAGHAD) = TEMP6
  328             DIFFER(TEMP2,'')                :F(FRETURN)S(RETURN)
      * 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 = LINSPC(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 = LINOVR(TEMP1)
  341 PRINT5      DIFFER(TEMP2,'')                :F(PRINTER)
  342             TEMP4 = IDENT(OVRASC(TEMP2),'N') OVRLST(OVRSTR(TEMP2),
     +                    'N',TEMP4)              :S(PRINT6)
  343             OUTPUT = OVRSTR(TEMP2)
  344 PRINT6      TEMP2 = NXTOVR(TEMP2)           :(PRINT5)
  345 END.PRINTER
      *
      *-----------------------------------------------------------------------
      * Print pending overprints
      * The input outpu 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

Decode text commands.
      * Hexadecimal character insertion pattern definition
      *
  352             HEXER = ANY('0123456789ABCDEF') . I1
      +                   ANY('0123456789ABCDEF') . I2
      *
  353       DEFINE('ESCAPEFN(CENCMD)')          :(END.ESCAPEFN)
      *
      * Test if hexadecimal inclusion command
      *
  354 ESCAPEFN    ESCAPEFN =                      ;* set to null string
  355             CARD HEXER =                    :S(HEXINC)
      *
      * Test if it is an underscore command
      *
  356             CARD ANY(UNDCMD) . I1 =         :S(ISUNDER)
      *
      * Test if it is a single overprint command
      *
  357             CARD ANY(SOVCMD) . I1 =         :S(ISSINOV)
      *
      * Test if word break commend
      *
  358             CARD ANY(SEPCMD) . I1 =         :S(SEPAR)
      *
      * Test if word or line fill command
      *
  359             CARD ANY(FILCMD) . I1 =         :S(ISFILL)
      *
      * Test if any cent function command
      *
  360             CARD ANY(CENCMD) . I1 =         :F(RETURN)
  361             I1 ANY('PZIUT/')                :F($('CENT' I1))
  362             ESCAPE = ESCAPE I1              :(FRETURN)
```

```
A text formatter, version 2

Overprint string generating functions.
      * Underscore handling function
      *
      * Test if underscore was in progression. USCFLG tells that.
      *
  363 ISUNDER     GT(USCFLG,0)                    :F(UNDLOOP1)
      *
      * 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 USCLSTPT, USCFLG and USCPTR.
      *
  364             USCSTR(USCLSTPT) = DUPL(USCSTR(USCLSTPT), SIZE(WRD) -
     +                    USCPOS(USCLSTPT))
  365             ESCAPE BREAK('u') . I2 'u' = I2         ;* no further
      *
      * Verify if this command initiates a continuous underscore
      *
  366 UNDLOOP1    USCFLG = 1                      ;* set to non continuous
  367 UNDLOOP2    IDENT(I1,'&')                   :F(UNDLOOP3)
  368             USCFLG = 2                      ;* set to continuous
  369             CARD ANY(UNDCMD) . I1 =       :S(UNDLOOP2)F(UNDLOOP7)
      *
      * Test if underscore end command
      *
  370 UNDLOOP3    IDENT(I1,'.')                   :F(UNDLOOP5)
  371 UNDLOOP7    USCFLG = 0                      ;* no more underscore
  372             USCLSTPT =                      :(RETURN)
      *
      * Obtain Underscore descriptor pointer and build an underscore entry
      *
  373 UNDLOOP5    USCPTR = $('ZUSC' I1)           ;* get pointer
  374             USCLSTHD = USCLST(ULNSYM(USCPTR),SIZE(WRD),ULNASC(USCPTR),
     +                    USCLSTHD)
  375             USCLSTPT = USCLSTHD
  376             ESCAPE = ESCAPE 'u'             :(RETURN)
      *
      * Processing of end of word underscore alarm
      *
  377 ESCAPu      USCSTR(USCLSTPT) = DUPL(USCSTR(USCLSTPT),SIZE(WRD) -
     +                    USCPOS(USCLSTPT))
  378             USCLSTHD = EQ(USCFLG,2) USCLST(ULNSYM(USCPTR),0,
     +                    ULNASC(USCPTR),USCLSTHD)    :F(UNDLOOP6)
  379             USCLSTPT = USCLSTHD
  380             ESCTMP = ESCTMP 'u'             :(ESCAPEND)
      *
      * There are no further underscores
      *
  381 UNDLOOP6    USCFLG = 0
  382             USCLSTPT =                      :(ESCAPEND)
      *
      *-----------------------------------------------------------------
      * Single character over print handling
      *
  383 ISSINOV     SINPTR = $('ZSIN' I1)
  384             USCLSTHD = USCLST(ULNSYM(SINPTR),SIZE(WRD) - 1,
     +                    ULNASC(SINPTR),USCLSTHD)        :(RETURN)
      *
      *-----------------------------------------------------------------
      * Word separating command handling.
      *
  385 SEPAR       SINPTR = $('ZSEP' I1)
  386             SEPLSTHD = USCLST(ULNSYM(SINPTR),SIZE(WRD),'',SEPLSTHD)
  387             ESCAPE = ESCAPE 's'             :(RETURN)
      *
      *-----------------------------------------------------------------
      * Long over print command handling
      *
  388 CENT#       TEMP1 = 0                       ;* Base string flag
  389             TEMP2 = 'C'                     ;* current line association
```

```
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 '#' =                       :F(OVFLOOP1)
  401             TEMP2 = 'N'                      :(OVFLOOP1)
      *
      * Complete the case for ¢
      *
  402 OVFN¢       TEMP4 = TEMP4 I1 ESCAPEFN('¢V') :(OVFLOOP2)
```

A text formatter, version 2

Non delimitting cent commands.

```
      *
      * Hexadecimal inclusion
      *
  403 HEXINC      ESCAPEFN = SUBSTR(&ALPHABET,16 * CONV(I1) + CONV(I2) + 1,1)
      +                                          :(RETURN)
      *
      *-----------------------------------------------------------------------
      * Double cent command
      *
  404 CENT¢       ESCAPEFN = '¢'                 :(RETURN)
      *
      *-----------------------------------------------------------------------
      * Start a new card without starting a new line
      *
  405 CENTV       CARD =                         :(RETURN)
      *
      *-----------------------------------------------------------------------
      * Page parameter definition command ¢G
      *
  406 CENTG       CARD ('H' . I1 | ANY('123CDNRSZ#') . I1
      +                (SPAN('0123456789') | '') . I2 ) =
      +                                          :S($('CENTG' I1))F(RETURN)
  407 CENTGH      ESCAPE = ESCAPE 'G'            :(FRETURN)
  408 CENTGD      PAGSIZ = GT(I2,PAGINI) I2      :(RETURN)
  409 CENTG#      PAGENO = I2                    :(RETURN)
  410 CENTGN      FOTSTT = GT(I2,4) I2           :(RETURN)
  411 CENTG1      ;CENTG2      ;CENTG3
      +           $('FIGSP' I1) = I2 + 1         :(RETURN)
  414 CENTGR      SEQALN = I2                    :(RETURN)
  415 CENTGS      SPCING = I2                    :(RETURN)
  416 CENTGZ      SPCLIN = I2
  417             CARD 'R' =                     :F(RETURN)
  418             SPCLIN = SPCLIN + SPCING       :(RETURN)
  419 CENTGC      XY1 = I2                       ;* save value
  420 CENTGC1     GT(LINCNT + XY1,PAGSIZ)        :F(RETURN)
  421             IDENT(MINLIN,'A') PAGER(LINSAV)          :S(CENTGC1)
  422             LE(LINCNT,PAGSIZ) PAGER(LINCNT)          :S(CENTGC1)
  423             PAGER(LINCNT - MINEND)                   :(CENTGC1)
      *
      *-----------------------------------------------------------------------
      * String generating functions ¢J
      *
  424 CENTJ       CARD ANY('DSP#') . I1 =        :S($('CENTJ' I1))F(RETURN)
  425 CENTJD      ESCAPEFN = DATE()              :(RETURN)
  426 CENTJS      ESCAPEFN = EVAL(LINSTR(BUILDSTR()))              :(RETURN)
  427 CENTJP      :<CODE(LINSTR(BUILDSTR()) '; :(RETURN)')>
  428 CENTJ#      ESCAPEFN = PAGENO              :(RETURN)
      *
      *-----------------------------------------------------------------------
      * Page title reading command ¢H
      *
  429 CENTH       CARD ANY('AFMN0123456789') . SAVHAD =  :F(RETURN)
  430             TEMP1 = BUILDSTR()             ;* Obtain title string
  431             LINSTR(TEMP1) ' ' RTAB(1) . I1 ' ' = I1
  432             TITLE2 = IDENT(SAVHAD,'A') LINSTR(TEMP1)         :S(RETURN)
  433             FIGTIT = IDENT(SAVHAD,'F') LINSTR(TEMP1)         :S(RETURN)
  434             FOTTIT = IDENT(SAVHAD,'N') LINSTR(TEMP1)         :S(RETURN)
  435             FOTSTR = IDENT(SAVHAD,'M') LINSTR(TEMP1)         :S(RETURN)
  436             SAVHAD = GT(SAVHAD,PAGINI) PAGINI
      *
      * Loop through header lines until title line found
      *
  437             TEMP2 = LINLSTHD
  438             LINSTR(TEMP1) = EQ(SAVHAD,0) '1' LINSTR(TEMP1)   :S(CENTH2)
  439             LINSTR(TEMP1) = ' ' LINSTR(TEMP1)
  440 CENTH1      SAVHAD = GT(SAVHAD,0) SAVHAD - 1       :F(CENTH2)
  441             TEMP2 = NXTLIN(TEMP2)          :(CENTH1)
  442 CENTH2      LINSTR(TEMP2) = LINSTR(TEMP1)
```

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) = '  '
  446             LINOVR(TEMP2) =              :(CENTH3)
      *
      *-----------------------------------------------------------------------
      * Footnote command 'ǿN'. Obtain Footnote mark character or string
      *
  447 CENTN       FOTCNT = FOTCNT + 1
  448             ESCAPEFN = SUBSTR(FOTSTR,FOTCNT,1)
  449             CARD = ESCAPEFN ' ' CARD
      *
      * Collect footnote text
      *
  450             FOTIND = FIGFOT(ZIDNFOOT)
  451             ESCAPE = ESCAPE 'N'          :(RETURN)
      *
      *-----------------------------------------------------------------------
      * Collect figure text 'ǿS'. Surround it by delimmiting strings
      *
  452 CENTS       FIGIND = FIGFOT(ZIDNFIG)
  453             NXTLIN(FIGEND(FIGIND)) = LINLST(FIGTIT,FIGSP2)
  454             FIGEND(FIGIND) = NXTLIN(FIGEND(FIGIND))
  455             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)
      +                   TEMP1                :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 'ǿK'
      *
  470 CENTK       CARD ANY('IOUBF') . I1 BREAK('(') . I2 =          :F(RETURN)
  471             CARD (BREAK(')') ')') . I3 = I3 'ǿT '             :F(RETURN)
  472             LINSTR(BUILDSTR()) (SPAN(' ') | '') REM . TEMP1
      +                                 :S($('CENTK' I1))F(RETURN)
  473 CENTKI      $('ZIDN' I2) = EVAL('IDNDEF' 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)
```

```
Non delimitting cent commands.
  478 CENTKB      $('ZSEP' I2) = EVAL('ULNDEF' TEMP1)
  479             SEPCMD = SEPCMD I2                          :(RETURN)
  480 CENTKF      $('ZFIL' I2) = EVAL('ULNDEF' TEMP1)
  481             FILCMD = FILCMD I2                          :(RETURN)
      *
      *-----------------------------------------------------------------------
      * 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                     ;* allways on string start match
  497             &STLIMIT = 16000000
  498             SETEXIT(.ERRHND)                :(AROUND)
  499 ERRHND      SETEXIT(.ERRHND)                ;* reestablish error exit
  500             OUTPUT = ' **ERROR**' &ERRTYPE '-ST ' &LASTNO ' |'
     +                  CARD SEQN                :(CONTINUE)
  501 AROUND      &ERRLIMIT = 10                  ;* allow errors to be handeled
  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)
  510             NXTLIN(LINLSTHD) = LINLST('  ',1,,,LINLST('  ',1,,,PAGHAD))
  511             PAGINI = 3
  512             LINCNT = PAGINI
  513             PAGSAV = 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(1,16,68,4,1,18,0,5,1,23,0,5,0,18,0)
  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('_','C')
  525             ZUSCd = ULNDEF('_','C')
  526             ZUSCb = ULNDEF('_','N')
  527             UNDCMD = 'ndb&.'
      *
      * Single overprint descriptors
      *
  528             ZSINa = ULNDEF('´','C')
  529             ZSING = ULNDEF('`','C')
  530             ZSINf = ULNDEF('^','C')
  531             ZSINt = ULNDEF('~','C')
  532             ZSINu = ULNDEF('¨','C')
  533             ZSINc = ULNDEF(',','C')
  534             ZSINs = ULNDEF('/','C')
  535             SOVCMD = 'agftucs'
      *
      * Word separator descriptors
      *
  536             $('ZSEP-') = ULNDEF('-','C')
  537             $('ZSEP,') = ULNDEF('','C')
  538             SEPCMD = '-,'
      *
```

```
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             FIGSP1 = 1
  543             FIGSP2 = 1
  544             FIGSP3 = 2
  545             FOTTIT = DUPL(' ',5) DUPL('-',20)
  546             FOTSTT = 20
  547             FOTSTR = '†‡*12345'
      *
      * Initialize reading environment
      *
  548             FIND = FINDA
  549             CRDEND = ' '
  550             CENCMD = 'ØGHIJKNPSTUVYZ/#'
  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)                   :(LOOP)
  560 LAAP        OUTPUT = '1 =====   END OF RUN   ====='
  561 END
```

Index