Block Bros.
Data Centre

BB-HUGO

0 Introduction

This short report describes the re-implementation of the HUGO language for text composition at Block Brothers Data Centre as of May 1981. It is described in terms of the HUGO Language Manual and Report produced at the Text Editing and Photocomposition Branch of the federal Queen's Printers in August 1980. The sections of this report correspond to the chapters of that manual, and where a chapter or feature is not mentioned it can be assumed to be implemented as described in the manual. References of the form (in 1.1.2) are to the chapters and sections of the HUGO manual.

The current implementation commenced in February 1981 and is still in progress. A different method of implementation has been used in this re-implementation. The HUGO program is translated directly into PL/I. This affords the advantages of greater speed and smaller size in running HUGO programs, and eliminates the need to re-compile programs at each use. This implementation can also support a variety of output devices. At present, a HUGO program can, by appropriate parameterization at run-time, produce output for a III VideoComp 570 phototypesetter, and for an IBM 3800 printer with both normal and "side-ways" character sets.

This is as yet only a partial implementation of HUGO. The major areas in which features are missing are support of various types of paragraphs, hyphenation of words to determine line breaks, and creation of source program segments. (HUGO programs never got big enough for this latter feature to be useful anyway.) Most other features not yet available were originally implemented to satisfy specific installation needs, such as accounting support, and are not very high on the requirements list for this implementation.

A number of new facilities and extensions of existing facilities have been included in the implemention when they were considered useful. In addition a number of features described in the manual, but never implemented in the original version of HUGO, have been included, such as the use of arbitrary strings with USER-CODES.

1 Basic Concepts

Numbers are stored internally either as whole numbers, with no fractional part, or scaled by a factor of 65536. While this allows more precision than the scaling by 1000 of the original implementation, and allows different devices to be more easily described, it does slightly change the results of certain language functions, and reduces the maximum values that may be stored in numbers. The INTEGER data type has been introduced in chapter 4 to aid the user in the representation of large numbers and to help compensate for this drawback.

Some characters are not supported by the compiler except in strings, due to their absence on the available input devices and a their not having a representation in EBCDIC. The definitions of <quoted constant> (in 1.1.2) and of <letter> (in 1.2) are therefore restricted to:

<quoted constant> =
' { <character> } ' |
" { <character> } "

<letter> =
A | B | C | E | D | F | G | H | I |
J | K | L | M | N | O | P | Q | R |
S | T | U | V | W | X | Y | Z

The only restriction on what may be used as a character is still that in 1.1.2. So " may only be used in strings delimited by ' and vica-versa.

To help input non-standard characters, a method of describing character strings in hexidecimal has been

added:

<quoted constant> = # { <hex digit> <hex digit> }

<hex digit> =
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
8 | 9 | A | B | C | D | E | F

The character string described consists of the characters represented by each pair of hex digits.

3 Statements

A file accessed by the PUT statement or the GET function can be reset so that the next access will be at the start of the file.

<statement> = RESET <expression>

The value of the expression is used as the name of a file. If the next operation after the RESET on the file is a GET, then that GET will reread the first record of the file. If the next operation is a PUT, then the old contents of the file are lost and replaced by that and following PUTS.

4 Declarations

The symbols STATIC and ENTRY are not supported (in 4.1.1, 4.1.2, 4.1.3, 4.2 and 4.3), together with the language facilities they support.

In addition to the types STRING and NUMBER, the type INTEGER is available (in 4.4.1):

<type> = STRING | NUMBER | INTEGER

An INTEGER object is a whole number.

Two new functions are available for forcing an object to be of type INTEGER (in 4.4.2):

<built-in function call> =
INTEGER (<expression>) |
ROUND (<expression>)

INTEGER has the same definition as the FLOOR built-in function (in 2.3.1). ROUND rounds the number to the nearest whole number instead of truncating it as do FLOOR and INTEGER.

5 Programs

 is not supported (in 5.1 and 5.3).

<at-job-start section>, <at-job-end section>, <at-page-start section> and <at-page-end section> are not supported (in 5.2 and 5.2.2).

6 Characters

Due to the different input and output devices available, the predefined input sequences (in 6.1.2) are not the same as in the original HUGO implementation, and may vary depending on the output device selected. In particular, no character is as yet defined as the "backspace" character. This means that the examples in the HUGO manual may not apply to this implementation.

Note that a predefined sequence or user-code (in 6.1.3) may be any sequence of characters, not just single characters or backspaced pairs.

The available fonts are different on different devices. Therefore the following character format statements (in 6.2.1) and built-in functions (in 6.4.2) are not supported: TIMES, MODERN, EXCELSIOR,

HEVETICA, OLD-HELVETICA, PERMA and BEDFORD. To replace them, the following definitions have been added:

```
<character format statement> =
        OPTIMA <expression> [, <expression> ] |
        PALATINO <expression> [, <expression> ] |
        CATALOGUE <expression> [, <expression> ] |
        OPTIMA-REVERSED <expression> [, <expression> ] |
        PALATINO-REVERSED <expression> [, <expression> ]

<Boolean built-in function call> =
        OPTIMA |
        PALATINO |
        CATALOGUE |
        OPTIMA-REVERSED |
        PALATINO-REVERSED
```

Two new sizing operators have been added to those in 6.4.1:

```
<subfactor> =
            <subfactor> RSUS |
            <subfactor> CENTIGRADE
```

An RSU is one 65536th of a point. RSU is a synonym for RSUS. One CENTIGRADE is one one-hundredth of a circle, or 3.6 degrees of angle. It can be used as the argument of the SLANT statement.

One of the output devices available with this implementation supports the display of halftone pictures or line drawings. Therefore a new statement has been added:

```
<character format statement> =
        LOGO <expression>
                [, <expression> [, <expression> ] ]
```

The first expression is the name of the required halftone or drawing. The optional second and third expressions provide scaling information if the picture is to be displayed in a size differing from the one in which it is stored or if its width is significant in placing text with it in a line. When present, the second expression specifies the horizontal size (or width) of the picture, and the third is the vertical size (or height). If the second argument is present but the third is not, the second is used for both values. If neither the width and height are given a zero value is used. The current text set-height or LEADING value is used for inter-line spacing, rather than the given height.

A new alternate form of the QUAD statement is available. It differs from the QUAD statement in allowing the vertical extent of the generated quad space to be specified. A QUAD-BLOCK may therefore affect the spacing between the line on which it is set and the preceding and following lines.

```
<character format statement> =
QUAD-BLOCK <expression> [, <expression> [, <expression> ] ]
```

As with the QUAD statement, the first expression specifies the width of the quad space to be generated. The optional second expression specifies the amount of space to be left above the current baseline for this space, and the optional third expression specifies the amount of space to be left below the current baseline. The interline spacing will be increased if necessary to accomodate these two space factors. If the second expression is not given then it defaults to the same value as the first. If the third expression is not given then it defaults to zero.

An alternate form of the QUAD-TO statement is also available:

```
<character format statement> =
QUAD-BEYOND <expression> [,<expression> ]
```

The QUAD-TO statement will go to the given position in a line even if text filling has already got beyond that point. This is not always the desired action. So the QUAD-BEYOND statement will QUAD the amount given by the second expression (which defaults to zero), and then attempts to do a QUAD-TO. If the QUAD has put it beyond the position in the line given by the first expression the QUAD-TO part of the action will be ignored.

7 Lines

The QUAD-WITH statement (in 7.3.2) can have an arbitrary string as its first argument, rather than just a single character. So filling can be done with any string. This is extended so that if any of the characters or sequences in the string invoke a USER-CODE that sets a FIGURE or a LOGO, these also are used as part of the filler. All or any part of the filler may also be underlined.

The SB-RATIO statement has been extended so that both a minimal and an optimal space-band size can be specified:

<line format statement> =
    SB-RATIO <expression> [, <expression>]

If only one expression is given in the SB-RATIO statement, then it acts as described in the manual. If two expressions are given, then the first is taken to be the minimum space-band ratio allowed, and the second is the desired space-band ratio. The effect of the optimal ratio will not normally be noticed in justified text, but in FLUSH-LEFT, FLUSH-RIGHT or CENTRED lines the optimum size will be used rather than the minimum.

"Dash" characters (hyphens, em and en dashes) normally indicate a valid place to break text (following the dash). A new statement, NO-DASH-BREAK, prevents text breaking following dashes, and the DASH-BREAK statement restores the normal condition.

<line format statement> =
        DASH-BREAK |
        NO-DASH-BREAK

The FLUSH-LEFT-WITH and JUSTIFY-WITH statements (in 7.3.2) are not supported, nor is the DISCRETIONARY statement (in 7.4) or hyphenation (in 7.5 and 7.7).

8 Paragraphs

Most of the paragraph make-up features are as yet unimplemented. The only statements in this chapter which are fully supported are SAVE, RESTORE, SAVE-DEFAULTS and RESTORE-DEFAULTS (in 8.1.2), and ON, LEADING, PARA-SPACE, NO-PARA-SPACE and SPACE-BLOCK (in 8.2.3), together with the PARA-SPACE and NO-PARA-SPACE built-in functions (in 8.2.4).

The SUB-PARA statement (in 6.2.1) and the HEAD, TAG, INTER-PARA and TEXT statements (in 8.3.1) are supported only to the extent of their effect on line indentation, line justification, and the saving and restoring of default attributes.

9 Pages

Only the page layout definition facilities in 9.1 and 9.2.4, the RESERVE, NEW-BODY, FINISH-BODY and FINISH statements (in 9.2.1) and the built-in functions AT-Y, MAX-DEPTH and MAX-WIDTH (in 9.5.1) and A0, A1, A2, A3, A4, A5, A6, A7 and A8 (in 9.5.3) are supported.

Galley-form make-up has been extended to allow the RESERVE and NEW-BODY statements in GALLEYS, but with the same restrictions as given for FINISH-BODY and FINISH in 9.2.4.

An AT-X built-in function has been added with a meaning analogous to that of the AT-Y function:

<built-in function call> = AT-X

It returns the first argument of the most recent AT statement encountered.

10 Ruling and Underlining

The following forms of ruling are supported: underlining (in 10.1.1), rule filling (in 10.2), figure rules (in 10.3), and ruling in a page layout (in 10.4). All the related built-in functions are supported (in 10.5). Side-lining (in 10.1.2), FLUSH-LEFT-WITH-RULE and JUSTIFY-WITH-RULE (in 10.2), and ruling in the page body (in 10.5) are not supported.

11 Text Input

A new builtin function is available:

<built-in function call> = GET-NEXT-LINE

This function will return the next line from the same file that is being used by the IF-YOU-FINDS. That line is thereafter deleted from the input and will not be encountered by the pattern matching facilities of HUGO. This function can be called repeatedly and will input successive lines from the input, returning the null string when and if it encounters the end of the input file (similar to GET).

12 General Enquiries

The name of the SECONDS built-in function has been changed to MINUTES with the obvious change in meaning.

Appendix A Implementation Dependent Features

The SET-ABSOLUTE statement has been redefined. It now functions exactly as does the SET statement, except that no USER-CODES or ESCAPE-CHARS are recognized in the string being set. This allows the pre-defined meaning of an input character or sequence to be accessed even though it is being used as a USER-CODE or ESCAPE-CHAR.

The ESCAPE-CHAR feature has been extended so that the ESCAPE-CHAR can be a string of one or more characters.

Documents and their related functions (all of A.3) are not supported.

Job accounting is not supported (in A.4), but the JOB-NAME built-in function has been implemented. Two new functions, OUTPUT-DEVICE and PARM, are supported:

<built-in function call> =
OUTPUT-DEVICE |
PARM

OUTPUT-DEVICE returns a string which is the name of the device for which HUGO is preparing set data. The devices now supported are "VC570", "3800-BLST", "3800-ET69" and "3800-SCS0". These are described in more detail later. PARM returns the value of the parameter supplied to the HUGO program at run time using the PARM field on the JCL EXEC card. Conforming to PL/I conventions, the parameter supplied must be preceded by a slash.

The ERROR and WARNING statements (in A.6) are supported, but the ERROR built-in function is not.

Appendix B Language Summary

There are different default attributes (in B.4) for each of the supported output devices. Where they differ they are given together with the detailed descriptions of the available fonts at the end of this report.

Appendix C Using the TEPB Implementation of HUGO

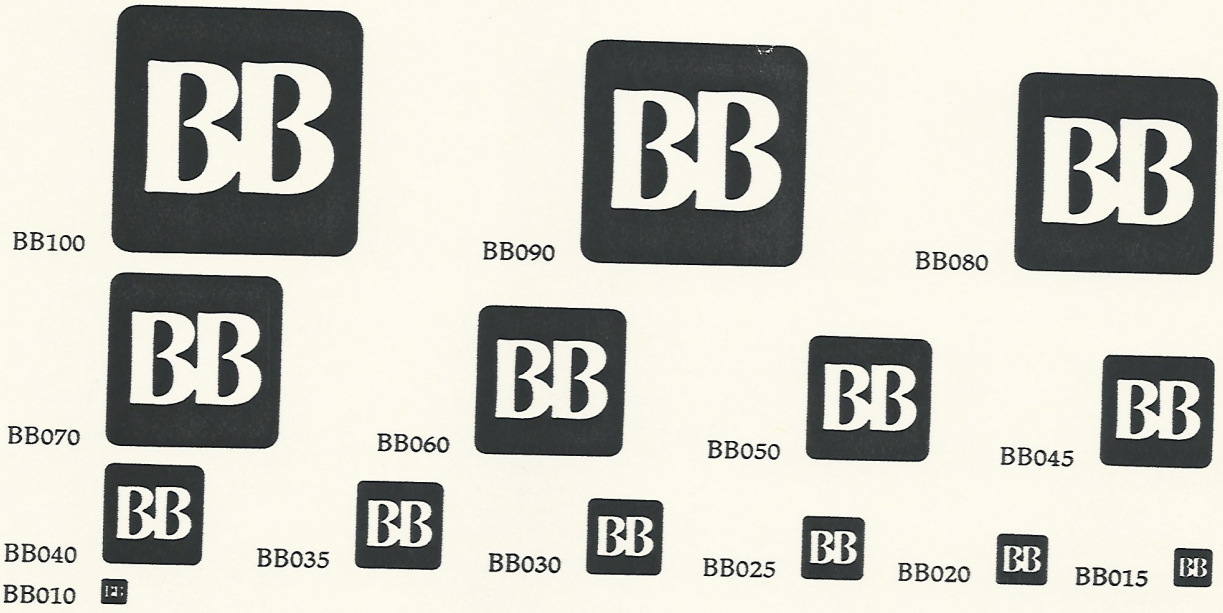
The control cards and options for running the current HUGO implementation are quite different from those used at TEPB. For the present, the JCL in the library BL.HUGO.JCL in members RUN570 and RUNSCS0 should be used as models for jobs for each of the devices.

Appendix D Font and Character Access

This is, of course, where the current implementation differs most from the original one. The following describes the available LOGOs and character sets, and the defaults attributes available on each device.

D.1 LOGOs

It is expected that users of HUGO will create their own LOGOs and keep them in their own libraries. However, a number of generally useful symbols will be kept in the data set BL.HUGO.LOGO. At present it contains the following sizes of BB symbols. In each case the three digits in the name of the LOGO are the approximate width and height of the symbol.



D.2 Default Attributes

The default attributes are generally those described in the HUGO manual. The only exceptions at present are the default font, setsize and line width on each device. The ruling and underlining commands are not supported on the IBM 3800, and will be simply ignored. The defaults for each device are:

For the VC570

FONT 100, 9 POINTS ON 10 POINTS
WIDTH 200 POINTS
SB-STEP 1 RSU

For the IBM 3800-ET69

FONT 0, (1 DIV 12) INCH BY (1 DIV 6) INCH
ON (1 DIV 12) INCH
WIDTH 7 INCHES
SB-STEP 1 EN

For the IBM 3800-SCS0

FONT 0, 0.1 INCH BY (1 DIV 6) INCH
ON 0.1 INCH
WIDTH 7 INCHES
SB-STEP 1 EN

For the IBM 3800-BLST

FONT 0, 0.125 INCH BY (2 DIV 15) INCH
ON 0.125 INCH
WIDTH 9 INCHES
SB-STEP 1 EN

D.3 Fonts and Character Sets

The following pages display the characters available in each font on each supported device. The only predefined input sequences at present are single characters, and the tables indicate the output resulting from each character in the EBCDIC set. Positions left blank are undefined, except that the "space" character (hexidecimal 40) results in the SPACE-BAND action.

The relationship between the fonts on the VC570 is:

Font 100 is Palatino.
Font 105 is Palatino italic.
Font 110 is Palatino bold.
Font 111 is Palatino-reversed.
Font 200 is Optima.
Font 205 is Optima italic.
Font 210 is Optima bold.
Font 211 is Optima-reversed.
Font 315 is Catalogue.
Font 415 is Catalogue bold.

ot sup

70
100, 9 POINTS
200 POINTS
SB-STEP 1 RSU

For the IBM 3800-ET69