CypherText: An extensible composing and typesetting language

by C. G. MOORE and R. P. MANN

The Cyphernetics Corporation
Ann Arbor, Michigan

INTRODUCTION

CypherText is a programming language designed for text formatting and typesetting in a time-sharing environment. Text to be formatted or typeset is input on a terminal and may be output at the terminal or on various typesetting machines.

Although a number of computer typesetting languages have been written for particular applications (such as newspaper work), few of these languages are adaptable for any other application.[1,2,3] This inflexibility has remained one of the most serious limitations of existing computer typesetting languages.

CypherText, an extensible language, overcomes this problem of inflexibility to a great extent. Because CypherText is truly extensible, it is possible to tailor specific formatting capabilities to meet the needs of particular typesetting applications by predefining formats for each application. Both large scale projects such as catalogs and parts lists, as well as smaller operations, such as job-shop typesetting, may now be accommodated within the scope of one language.

By predefining formats, a set of format definitions for a specific application may be "packaged" so that the definitions come "ready to use," i.e., the user does not have to know anything about how to make up formatting definitions for himself. This "packaging" of formats has already been accomplished for architectural specifications, technical report writing, and job-shop typesetting applications. In the first two cases, the format definitions are so comprehensive that the user almost never requires any of the unextended features of the language. In fact, most users are unaware of the "unpackaged" features because the packaged definitions meet all their formatting requirements.

In addition to providing wide formatting flexibility, CypherText also provides flexibility in choosing typesetting devices on which the text is to be output. Other typesetting languages have typically been geared to one or a few specific typesetting machines. CypherText, on the other hand, is "device independent": a "post-processing" feature allows users to set their text on many commercially available typesetting devices, including photocomposition devices, "hot lead" devices, and even typewriter-like terminals, with no change required in the input text.

The extensibility of CypherText and the flexibility it offers derive from the structure of the language and the method of its use.

THE STRUCTURE OF CYPHERTEXT

The major structural features of CypherText are its command syntax, command definition capability, and string storage capability.

Syntax

One prerequisite of an extensible typesetting language is an unambiguous syntax. Every effort has been made to keep the CypherText syntax simple and consistent.

CypherText input consists of the text to be typeset and the formatting instructions for the text. The formatting instructions ("commands") are distinguished from the text by "command break characters." Though the command break character may be any character the user chooses, throughout this paper the slash (/) will be used. The following fragment of input shows some text and one command:

as shown on the following page/NEXTPAGE/

In the above example, the text "as shown on the following page" would be set on a particular page, after which the command "NEXTPAGE" would cause any subsequent text to be set on the next page.

More than one command may be placed within the

From the collection of the Computer History Museum (www.computerhistory.org)

break characters, provided that the individual commands are separated by semicolons, as in the following example:

as shown on the following page./NEXTPAGE;

CENTER/Chapter VI

In this example, the two commands "NEXTPAGE" and "CENTER" are placed within the same set of slashes; "NEXTPAGE" causes a skip to the next page, after which "CENTER" causes the text "Chapter VI" to be centered at the top of the new page.

Some commands require one or more modifiers (parameters) to fulfill their formatting functions. In these commands, the parameters are separated from the name of the command by a space, and multiple parameters are separated from each other by commas. For example, the command "SPACE" requires as a modifier the amount of vertical space to be left on a page, expressed in points. Thus, the command

/SPACE 24/

causes a vertical spacing of 24 points.

Among the commands requiring multiple parameters is "NEWTYPE", which has as modifiers the name, style, and point size of the type face to be set. Thus,

/NEWTYPE TIMES ROMAN,8/

would cause a switch to 8 point Times Roman as the current type face.

A list of the most commonly used CypherText commands and their functions is provided in Table I.

Command definition

The capability of defining new commands is integral to the extensibility of CypherText and contributes greatly to its ease of use.

New commands are created by combining a number of basic commands and assigning a name to the combination. The name is assigned by means of the "DEFINE" command. In the following example, a new command called "PT", requiring one parameter, "LINES", has been defined. The definition of the command appears between the quotation marks, and consists of three basic commands: SKIPIF, NEXTPAGE, and ENDIF:

/DEFINE PT(LINES),

"SKIPIF<,72, LINES *12;

NEXTPAGE;ENDIF"/

Having defined the new command "PT(LINES)", it would be used by supplying a value for the parameter

TABLE I—Commands

DEFINE
Used to define a new CypherText command. It gives a name to the command and indicates how the parameters are to be used.

ENDIF
(See SKIPIF)

EVALUATE
Evaluates an arithmetic expression and stores the value in a specified string name.

INCLUDE
Requests that the contents of some string (or combination of strings) be set as text at this point.

LEADER
Requests that a 'leader' of some particular character be used to fill out the current line of text. Used mostly in tables.

MAP
Gives a character whose occurrence is to be 'mapped' into some string. Every subsequent appearance of the mapped character will be treated as though the string of characters it is mapped into had occurred instead.

NEXTPAGE
NEXTPARAGRAPH
NEXTFIELD
Cause a new page, paragraph, or field (respectively) to be started at this point.

OUTPUT
Specifies the output device to be used for setting the text (for example, LINOFILM, PHOTON 713, terminal, etc.)

PUSH
POP
Together, allow the current contents of some string to be saved, and then later recovered.

SET
Assigns a new value to some string. Corresponds to the use of the equal sign (=) or replacement operator in most programming languages.

SKIPIF
Allows commands and text to be skipped (or ignored) in the setting process, if a specified condition is met. No text will be set or commands processed until an ENDIF command is encountered.

SPACE
Leaves a vertical space of the specified amount, on the page currently being composed, at the point the command occurs.

USE
Gives the name(s) of one or more files whose contents are to be included as input at this point. These files may include commands or text or both.

From the collection of the Computer History Museum (www.computerhistory.org)

"LINES". For example, the command

/PT 15/

would cause any text following the command to be set on the current page if more than 15 lines remain, or to be set on the next page if less than 15 lines remain.

String storage

The capability of assigning names to strings of characters and of storing the strings for future use is also crucial to the extensibility of CypherText. Strings are assigned names and stored for retrieval by means of the "SET" command. For example, the command

/SET X, "NEXTPAGE"/

would store the 8-character string "NEXTPAGE" under the name "X".

Such stored strings may be used as commands, parameters to commands, or even as text to be set. For example, after the above command, the command "X" is equivalent to the command "NEXTPAGE."

Using stored strings as commands or as parameters to commands merely involves substituting the string name for the string. For example, the sequence

/SET LINE, "12"/

.
.
.

/SPACE LINE/

stores the value "12" under the name "LINE", so that when "LINE" is used as a parameter to the "SPACE" command, a vertical spacing of 12 points is left on the page.

Using stored strings as text to be set involves the use of the "abbreviation character." Though this character may be any that the user chooses, in the following example the "at sign" (@) has been used.

Commonly used words, phrases, or paragraphs may be assigned string names and stored; whenever the words, phrases, or paragraphs are to be used as text, only the string name need be used, preceded by the abbreviation character. For example, the command

SET CT, "CypherText,

an extensible language,"

would store the quoted text under the name "CT". Whenever the user wants to include the text "CypherText, an extensible language," he has only to type in "@CT". In this case, 35 characters have been reduced to the 2-character abbreviation "CT".

Stored strings may also be used in a way analogous to the use of variables in algebraic programming languages. Thus, stored strings may be used in an arithmetic expression, as a parameter to a command. For example, the sequence

/SET LINES, "12"/

.
.
.

/SPACE 5*LINES/

causes a vertical space of 5 times the number of points specified in the string named "LINES" to be left on the current page.

Reserved String Names

Many of the formatting functions of CypherText are controlled by the use of "reserved string names". These are string names whose contents are constantly monitored by CypherText. Whenever the value of one of these reserved strings is changed, CypherText takes some special action. For example, the reserved string variable "LINELEADING" indicates the amount of space to be left between each line of the final text. Changing the value of this string will change the amount of space left between lines. Thus, the command

/SET LINELEADING,"12"/

indicates that from this point on, 12 points of space are to be left between each line in the output text. For typewriter-like terminals, this command is effectively a double-space command. As another example, the formatting of the top and bottom of each page is controlled by two reserved string names, "HEADER" and "TRAILER". Any combination of commands and text may be stored in these strings. Whenever CypherText begins a new page of text, it examines the contents of these strings to determine what to place at the top and bottom of each page. For example, the command

/SET HEADER,"/SPACE 36;CENTER;INCLUDE

TTEXT; SPACE 36/"/

stores in the reserved string "HEADER" a set of commands which will center at the top of each page the current contents of the string named "TTEXT", with 36 points of space between this line and the top of the page, and 36 points of space between this line and the first line of text. Of course, the contents of the string "TTEXT" may be changed at any time, via the "SET" command. Thereafter, each page will have the new contents of "TTEXT" as a centered title.

From the collection of the Computer History Museum (www.computerhistory.org)

TABLE II—Reserved Variables

FIELD
Controls the number, width, and placement of columns on the page. Also controls the placement of text within the field: centered, justified, flush right, flush left.

HEADER
Controls the formatting of the top of each page. Title, if any, spacing, and so forth.

HYPHENATION
Controls automatic hyphenation, which is done only if the contents of this string name is "ON".

INDENT
Specifies the amount of the indentation at the beginning of each paragraph.

JUSTIFICATION
Controls the amount of 'filling' with spaces allowed to justify a line of text.

LINELEADING
Controls the amount of space to be left between lines.

PAGEHEIGHT
Controls the height of each page.

PAGEWIDTH
Controls the width of each page.

PARAGRAPHLEADING
Controls the space to be left between each paragraph.

TRAILER
Controls the formatting of the bottom of each page, as with HEADER, at the top.

TYPEFACE
Controls the current type face (TIMES, BODONI, etc.).

TYPESIZE
Controls the current type size.

TYPESTYLE
Controls the current type style (ITALIC, BOLD, etc.).

A list of the most commonly used reserved string names and their functions is given in Table II.

Defined String Names

"Defined string names" is another class of string names which has special meaning to CypherText. These are strings which the user may always assume to contain some particular piece of information. Whenever the user references one of the defined string names, CypherText determines the current value of that piece of information and supplies that value as the value of the string. For example, the defined string name "TOTALPAGES" always contains the number of pages set so far during a particular run. The value stored in "TOTALPAGES" may be conveniently used to set a page number on each page.

Many of the defined string names are used primarily for testing certain conditions. The defined string name PAGELEFT contains the number of points (vertically) left on the current page, before it will be necessary to start a new page. Before beginning the setting of a table in his input text, a user may embed a conditional skip command ("SKIPIF") in his input which will test PAGELEFT to determine if there is enough room on the current page for the entire table. If there is not enough room, CypherText will start a new page; if there is enough room, the table will be set on the current page.

A list of the most commonly used defined string names and their functions is given in Table III.

Many other features of CypherText, such as automatic justification and hyphenation, are not discussed here because they are available in other languages as well.[4,5,6] The primary emphasis here has been to illustrate the extensible features of CypherText, particularly those features which differentiate it from other typesetting languages.

USING CYPHERTEXT

Using CypherText to transform original copy into finished text is a five step process:

1. Embedding
2. Inputting
3. Proofing
4. Postprocessing
5. Typesetting

Embedding is the insertion of CypherText commands into the original text. The commands may be written in by an editor for later inputting by a typist, or, in the case of experienced users, the commands may be embedded extemporaneously as the text is being input. The following example shows an original manuscript with the commands embedded by an editor:

```
/center/
/newtype HELVETICA, BOLD, 12/
CYPHERTEXT: A DEMONSTRATION
/space 12; newtype TIMES, ROMAN, 10; text/
#CypherText enables you to transform unformatted rough copy
into finished text by embedding CypherText commands in the
rough copy.
#The CypherText commands provide for all the formatting
requirements of the printed page, including justification,
hyphenation, tabulation, leadering, and runarounds.
```

From the collection of the Computer History Museum (www.computerhistory.org)

In this example, the commands specify that the heading is to be centered and set in 12 point Helvetica Bold, while the two paragraphs are to be set in 10 point Times Roman. (Note that the command for starting a new paragraph has been abbreviated to "#".)

After the commands have been inserted, the "embedded copy" is input into a general purpose time-sharing system on virtually any terminal input device.

Several advantages derive from the fact that the copy is entered into a time-sharing environment: first, the copy may be stored on any of a number of direct-access devices, depending on factors of economy and convenience; second, output from other programs in the time-sharing system may serve as input to CypherText; and third, the copy is always immediately accessible for updating.

The following example shows how the embedded copy would appear on a terminal during inputting:

```
/center/
/NEWTYPE HELVETICA,BOLD ,12/CYPHERTEXT%: A DEMONSTRATION
/space 12;NEWTYPE TIMES,ROMAN,10;TEXT/
#CypherText enables you to transform unformatted
 rough copy into finished text by embedding
 CypherText commands in the rough copy.
#The CypherText commands provide for all the
 formatting requirements of the printed page,
 including justification, hyphenation, tabulation,
 leadering, and runarounds.
READY
```

After the copy has been input, immediate proofs may be obtained by having the system compose and print out the text at the terminal. Of course, the proof copy takes on the limitations of the terminal on which it is output. However, for many applications, the proof copy obtained at the terminal is satisfactory enough to serve as final output for reproduction by printing or other means. For these applications, where limited type variety and non-proportional spacing are of no concern, the proof copy is the end product of the CypherText process and the last two steps, postprocessing and typesetting, are omitted.

Whether or not the proof copy is the final output, proof copy is useful for checking the formatting and for catching typographical errors. If errors are found, or if the formatting is to be changed, it is a simple matter to edit the input copy by using any of the text-editing facilities of the time-sharing system. After the copy is edited, further proofs may be output until the user is satisfied that the text is composed as desired.

The following example shows proof copy obtained at a model 37 teletype terminal:

```
   CYPHERTEXT: A DEMONSTRATION
      CypherText enables you to transform unformatted
rough copy into finished text by embedding CypherText
commands in the rough copy.
      The CypherText commands provide for all the
formatting requirements of the printed page, including justi-
fication, hyphenation, tabulation, leadering, and runarounds.
```

For those applications where a variety of type faces and proportional spacing are important, the next step is to postprocess the input copy for setting on a particular typesetting device. Postprocessing is handled automatically by the system, producing a tape (paper or magnetic) to drive any of the most commonly used typesetting machines.

To achieve complete typesetting flexibility, the CypherText language has been made as "device independent" as possible. This independence has been achieved by defining the input language independently of the characteristics of any specific typesetting device; the output is targeted to an idealized typesetting device (which does not actually exist). Producing output for an actual typesetting device is the function of the post-processing program, which translates the device independent output to the particularities of the desired typesetting machine. Translating the copy for typesetting on different machines requires only a change in the "OUTPUT" command, which takes one parameter, the name of the desired typesetting machine. The "OUTPUT" command is also used for obtaining drafts at the terminal, typewriter-like devices being considered a special kind of typesetting machine. The device independent translators generally run in parallel with the CypherText language itself, as co-routines, effectively making the entire process a one-pass operation.

The final step, typesetting, consists of running the postprocessed tape on a particular typesetting machine to obtain finished, typeset copy. The number of type

TABLE III—Defined Variables

DATE	Current data, in the form DAY-MONTH-YEAR.
LINECHARACTERS	Current number of characters set so far on this line.
PAGEDOWN	How much text has been set on this page, i.e., how far 'down' the page text has been set.
PAGELEFT	How much space is left on the page, vertically, before it will be full.
PAGELINES	How many lines have been set on the page currently being composed.
TIME	Current time of day, in the form HH:MM (24-hour time).
TOTALPAGES	Total number of pages set so far in this run.

From the collection of the Computer History Museum (www.computerhistory.org)

faces and sizes, as well as the spacing characteristics, depend, of course, on the typesetting machine itself. The following example shows the sample text set on a Linofilm Quick:

CYPHERTEXT: A DEMONSTRATION

CypherText enables you to transform unformatted rough copy into finished text by embedding CypherText commands in the rough copy.

The CypherText commands provide for all the formatting requirements of the printed page, including justification, hyphenation, tabulation, leadering, and runarounds.

Although CypherText can be used in any composing and typesetting application, it is especially suited for text requiring frequent revision, complicated or repetitive formatting, and high speed and accuracy. Despite the sophisticated capabilities of the language, experience has shown that both novices and trained editors alike can be taught to use CypherText easily and effectively in a broad range of composing and typesetting applications.

APPENDIX A

SYNTAX OF THE LANGUAGE

Normally CyperText operates in "text mode", a mode in which the characters in the input stream are simply set according to whatever current formatting parameters are in effect. Commands which alter the formatting parameters may appear anywhere in the input text stream. These commands are bracketed by the current "command break character", which is normally a slash(/). One or more commands placed between command break characters in this manner is called a "command group", and must follow certain syntactical rules.

The syntax of a command group is given below. Rigor in the formal sense has been sacrificed for readability. Such sacrifices are indicated by enclosing parentheses. In the definitions we use the convention that lower case character strings stand for a generic type. Upper case strings and punctuation characters not mentioned in these conventions must appear as shown. Square brackets surround optional material. Three dots following a syntactic unit indicate that it may be repeated an arbitrary number of times. The sequence ':=' is used to mean 'is defined as'. A vertical bar is used to indicate that one of the options in curly brackets should be chosen. Curly brackets are also used to group syntactic units for some purpose. The special generic name 'nullstring' and 'blankstring' stand for a string of no characters and a string of one or more blank characters, respectively. The generic name 'alphanumericstring' stands for an arbitrary string of upper and lower case letters and numbers. The generic name 'numericstring' stands for a string of digits, possibly with a leading plus or minus sign, and an optional embedded decimal point.

commandgroup := commandbreakcharacter
commandstring
commandbreakcharacter

commandstring := [commandelement;]···
commandelement

commandelement := {primitivecommand |
macrocommand |
stringname | nullstring}

primitivecommand := (one of the commands from Table I) blankstring parameterlist

macrocommand := macroname blankstring parameterlist

stringname := alphanumericstring (of length less than 64 characters, beginning with a letter, and which has previously appeared as the first parameter of a SET or EVALUATE command).

macroname := alphanumericstring (of length less than 64 characters, beginning with a letter, and which has previously appeared as the first parameter of a DEFINE command).

parameterlist := [{(parameterlist) |
simpleparameter},]···
{(parameterlist) | simpleparameter
| nullstring}

simpleparameter := {alphanumericstring |
stringexpression |
numericexpression}

stringexpression := {stringname | quotedstring}
[& stringexpression]

quotedstring := "alphanumericstring" |
'alphanumericstring'}

numericexpression := {numericstring | stringname}
[arithmeticoperator
numericexpression]

arithmeticoperator := {+|−|*|/}

APPENDIX B

Implementation details

CypherText has been implemented for a PDP–10 time-sharing system. It is written entirely in assembly

From the collection of the Computer History Museum (www.computerhistory.org)

language. This choice was dictated by the fact that the only other option available at the time was FORTRAN. FORTRAN was felt to be too awkward and inefficient to use as an implementation language for what is essentially a string-handling program. It should be noted that higher-level languages available on other computers (such as PL/1) would be unquestionably preferable for implementing this type of program.

The programs, both first and second passes, are reentrant. In fact, the PDP–10 system allows the first pass to be shared simultaneously by a number of time-sharing system users. The first pass program occupies about 6500 (36-bit) words of memory for the code. A minimum of 6000 additional words are needed for working storage (page buffers, string storage, etc.).

The size of the second pass programs, which are usually loaded with the first pass for a particular run, varies considerably with the type-setting device selected. All the current second pass programs are less than 2000 words long, including both code and working storage.

The device independence of the PDP–10 input/output support allows input text to be accepted from a variety of media. The same comment applies to system output. No scratch files are written by the system, but CypherText does access several support files in the course of a run, which must be stored on a random access device.

The running time of the program varies with the number and complexity of the commands embedded in the text. For "straight matter", such as non-technical books, running time for first and second passes combined is about .3 seconds per 1000 characters. For very complicated work, such as some parts catalogs, run time may approach 2 seconds per 1000 characters. Unless final copy is being printed at the terminal, additional time will be needed on the type-setting device chosen to set the text.

REFERENCES

1 G M BERNS
Description of FORMAT, a text processing language
Comm of the ACM Vol 12 3 March 1969 pp 141-146

2 *TEXT360: Introduction and reference manual*
Form C35-0002 IBM Technical Publications Dept White Plains N Y March 1969

3 *Harris Composition System: Language manual*
Harris-Intertype Corp Cleveland Ohio March 1970

4 *Textran-2: User's manual*
Form T2-102-3 Alphanumeric Inc Lake Success N Y 1969

5 J W SEYBOLD
The market for computerized typesetting
Printing Industries of America Washington D C 1969

6 *HYPHENATION360: Application description*
Form E20-2130 IBM Technical Publications Dept White Plains N Y 1969

From the collection of the Computer History Museum (www.computerhistory.org)

From the collection of the Computer History Museum (www.computerhistory.org)