IEEE TRANSACTIONS ON

ENGINEERING WRITING AND SPEECH

AUGUST 1968 VOLUME EWS-11 NUMBER 2

Published Aperiodically

Special Issue on Computer-Aided Engineering Documentation

SCRIPT, An On-Line Manuscript Processing System

STUART E. MADNICK AND ALLEN MOULTON

Abstract—**The SCRIPT commands of the IBM CP67/CMS system provide interactive creation and editing of manuscript text, and can format and output hard copies. The primary goal of SCRIPT was a convenient method within an on-line system to permit programmers to prepare and maintain system documents. The SCRIPT commands have, however, also been used extensively to prepare technical reports and papers of all kinds.**

The EDIT module creates and operates upon a file in secondary storage. Lines of text from the typewriter terminal are put into canonical form and added to or inserted in the file. Editing instructions can reference a line either by context or relative line number, and can change strings within the line to other strings, as well as retype, delete, or insert whole lines.

The PRINT module formats the manuscript file and outputs it to either the typewriter or line printer. Lines are left and right justified by adding additional embedded blanks where necessary. A logical topology data structure correctly interprets overprinting characters. Format control lines may be inserted in the text to specify such additional features as heading lines, line length, page length, page numbering, centering, indentation, and double spacing. The design of a follow-on system with major changes in each of these areas is discussed (see Fig. 5 of the text).

I. INTRODUCTION

THE SCRIPT[1] system provides both on-line interactive editing and manuscript formatting. In earlier systems editing and formatting have usually been approached separately. Most systems for automatic formatting by computer, such as TEXT/90 [10], were designed for book publication and require specialized equipment and manually prepared input. Editing systems, on the other hand, have been directed primarily toward creating and modifying card-image program and data files, eliminating the inconvenience of handling and storing large numbers of punched cards. With the advent of time-sharing systems, elaborate file systems, and low-cost computers, there has been considerable activity in on-line editing. Early work was done by Samuel [3], Edwards [5], and Daley [8]. More recently, sophisticated techniques have been demonstrated by Murphy [1], Deutsch [2], and Sampson [12].

Manuscript received March 29, 1968; revised April 24, 1968.

S. E. Madnick is with the Sloan School of Management, Massachusetts Institute of Technology, Cambridge, and the Cambridge Scientific Center, IBM Corporation, Cambridge, Mass.

A. Moulton is with the Department of Political Science, Massachusetts Institute of Technology, Cambridge, and the Cambridge Scientific Center, IBM Corporation, Cambridge, Mass.

[1] The SCRIPT program is available only as a component of the CP67/CMS system, which is an IBM type III program.

There have been several attempts to combine editing and formatting, ranging from elaborate typewriter machines to large dedicated computer systems. The outstanding efforts have been Saltzer [7], Mathews [4], Zyrl [6], and Refs. [9], [11]. The design objectives of these combined edit-format programs, however, have not fully employed the technology demonstrated in the separate areas. The independent automated typesetting systems and on-line editing systems are designed for specialized users: in the first case, specially trained publications staff, and the latter, sophisticated programmers. Most of the combined editing–formatting systems are directed toward augmenting the secretary. In this they essentially mimic an elaborate typewriter mechanism and might be considered comparable to mechanical devices of such nature [15].

The SCRIPT system provides the nonclerical user of a general purpose time-sharing system, CP67/CMS [17] with the capabilities of a computer-based editing and formatting system. SCRIPT was originally designed to aid in up-to-date documentation of the time-sharing system itself. In use for more than a year, it has been found applicable to a wider range of problems. The time-sharing system is used by scientists and engineers, as well as applications and systems programmers. These users often wish to write, modify, or examine reports and documentation at arbitrary, unscheduled times. Without an on-line manuscript maintenance system many of the minor but valuable reports and modifications to documentation would not be made—because of the inconvenience or delay often encountered in secretarial schedules and document reproducing centers. SCRIPT is simple to use and powerful enough for the user to be willing to interrupt an on-going activity at the terminal, write or modify a document, and return to his other work.

In designing and constructing the SCRIPT system within the framework of an existing time-sharing system, it was necessary to use the technology from both editing and formatting to develop new techniques needed for the combined system. In particular, procedures were devised for efficient file access and text buffering, string processing, and placing input in canonical form. Equally important was the selection of powerful yet convenient command languages for editing the text and controlling the automatic formatting process; many ideas from Saltzer's TYPSET/RUNOFF were used in this area. Finally, a logical topology data structure was used to assist the formatting and effect the different requirements for output to the typewriter terminal or line printer.

II. SCRIPT System

System Description

The SCRIPT system consists of two modules, the "editor" and the "printer." The editor is used to create and operate upon a line-marked file from secondary storage. Lines of text from the typewriter terminal are put into canonical form and added to or inserted in the file. Edit instructions reference a line either by content or relative line number, and can change strings within the line to other strings, as well as retype, delete, or insert whole lines.

The printer formats the manuscript file under the direction of format commands in the text. The input text from the editor consists of free-format lines 1 to 132 characters in length. Under normal operation, "fill" mode, the text is rearranged by "spilling" words back and forth between the original input lines to balance the line length; additional embedded blanks are then added where necessary to left and right justify lines to a designated length. The format command lines specify additional features such as heading lines, line length, page numbering, centering, indentation, spacing, and others. The formatted output may be printed on either the user typewriter terminal or the high-speed line printer.

Entering Text from the Terminal

The editor accepts commands from the terminal and immediately acts upon them. A basic concept is the "current-line" pointer. The file may be considered a linear sequence of lines with a pointer which may be moved forward or backward within the file, but at each instance is positioned before, after, or on a specific line—the current line. The EDIT requests are of two types, those that operate on the current line and those that reposition the pointer.

The INPUT command is used to create a new file or insert text within an existing file immediately after the current line. It specifies that all further lines are to be treated as raw text. The pointer is adjusted to each new line as entered. A null line (two consecutive carriage returns) causes resumption of the editor command mode.

Typographical errors may be corrected on the line being typed by use of the "line erase" and "character erase" characters. The line erase character, normally the cent sign, specifies that all previous characters on the line are to be ignored. The character erase character, normally the commercial "at" sign, indicates that the immediately previous character is to be ignored. Multiple character erase characters may be used to delete a corresponding number of characters. When the carriage return is pressed, the line is edited to resolve the line erase and character erase characters.

Basic Editing Features

The REPLACE command deletes the current line and replaces it with the new line entered. The NEXT, PRINT, and DELETE commands move the pointer a designated number of lines forward. The PRINT command prints the lines as the pointer advances, the DELETE command deletes the lines as the pointer moves. The BOTTOM command moves the pointer to the last line of the file, prints the line, and enters INPUT mode. The TOP command places the pointer before the first line of the file.

The QUIT, FILE, and KEEP commands affect the entire file. The original file is not immediately altered during the editing process. The QUIT command terminates editing, leaving the original file intact and deleting the edited file. The FILE command terminates processing, but saves the newly edited file under the name designated; it may replace the original file or be saved as a separate file. The KEEP command is used to split the edited file into two or more separate files.

The EDIT command keywords, such as REPLACE, need not be typed in upper case; furthermore, they may be abbreviated as a single letter.

Context Editing Features

The CHANGE command specifies two character strings; the current line is searched for the first string, which is then replaced by the second. For example, if the current line is "Boston is the home of the bean and the cod," the command "CHANGE /cod/sheep/" will change the current line to "Boston is the home of the bean and the sheep."

The general form of the CHANGE command is:

CHANGE %string1%string2% (option)

where '%' is any nonblank character that does not occur within string 1 or string 2. The option allows specification of the scope of the CHANGE. Ordinarily only the first occurrence of string 1 is changed, but it is possible to designate that all occurrences on the current line, a specified number of following lines, or all following lines be altered.

Although the NEXT command allows positioning of the current line pointer on the basis of line number, this is often not convenient, since line numbers constantly change as the file is being edited. The SEEK and LOCATE commands reposition the pointer on the basis of the line's content rather than position. Both commands specify a character string. Starting from the current line, the pointer is advanced until a line containing the designated character string is found.

Verification of Editing

With manually prepared manuscripts, corrections may be made in a variety of ways, but eventually it becomes necessary to completely retype whole pages or the entire manuscript. This can be annoying, since new mistakes in spelling, punctuation, or omission often occur in retyping.

Ideally, on-line editing systems avoid this problem, since only the actual corrections need be typed. But it has been truthfully stated that "computers allow for bigger mistakes than ever conceived by humans." Inserting or deleting a line at the wrong place, for example, can occur via computer editing.

The SCRIPT system provides verification of editing in several ways. By designating "verify" mode, all alterations

by the CHANGE command are displayed at the typewriter terminal. The PRINT command allows inspection of the edited input text. Finally, one or more designated pages may be formatted and printed. Since the original file remains unchanged, if an "uncorrectable" mistake has been made, it is always possible to revert to the original and restart.

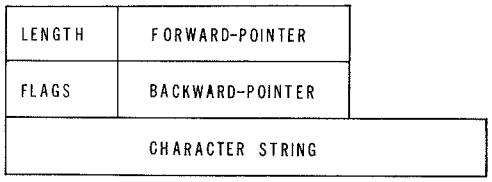


Fig. 1. Threaded-list element.

Text Storage and Buffering

The SCRIPT text file is kept on secondary storage as a blocked line-marked file. A line-marked file consists of sequences of line-length indicators followed by data. It is equivalent to a tape file consisting of variable-length records. The editing process involves three sequential disk files: 1) the original text file, 2) a scratch file, and 3) another scratch file. On the first pass, as the pointer is advanced through the file, input is read from file 1, edited and/or merged with input from the typewriter and written onto file 2. For the second, and all further passes, files 2 and 3 are alternately used for input and output. The line-marked file organization allows only forward sequential processing.

To reduce the input/output load for multi-pass editing and to allow moving the current-line pointer backward, text is buffered in main storage; approximately 160 000 bytes are at present available. Each line is made into an element of a two-way threaded list as illustrated in Fig. 1.

As lines are read from secondary storage or the terminal, they are added to the end of the list. When a deletion or insertion is made, only a few pointers within the list need be altered; see Fig. 2 for an example.

Text is written out only when there is insufficient storage to accommodate the new line read from the file or typewriter. Manuscripts of up to 20 pages in length can be processed without requiring intermediate input/output. For convenience, as well as efficiency, manuscripts longer than 20 pages can be subdivided into separate files, where each file is a logical entity such as a chapter.

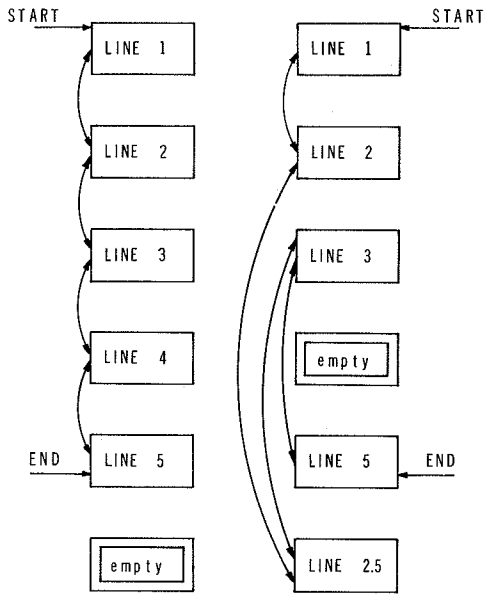


Fig. 2. Sample rearrangement of threaded list.

TABLE I

NEED FOR CANONICAL FORM

(The characters "←" represent backspaces.)
Target line: 1) SCRIPT← ← ← ← ← _ _ _ _ _
LOCATE character strings: 2) SCR← ← ← _ _ _
or: 3) S← _C← _R← _

Canonical Form of Input

Each new line entered from the typewriter is put into canonical form with respect to underlined and overprinted characters. The need for a canonical form is illustrated in Table I.

Often, underlined text is entered by typing the line, backspacing to the first character, and then typing the underscores. To locate a line by content, it is desirable to specify only the minimum unique character string. In Table I, each of the LOCATE character strings are logically contained on the target line, but physically might be stored as different character strings. To eliminate this difficulty, all lines are stored internally in the form of character string 3) in Table I.

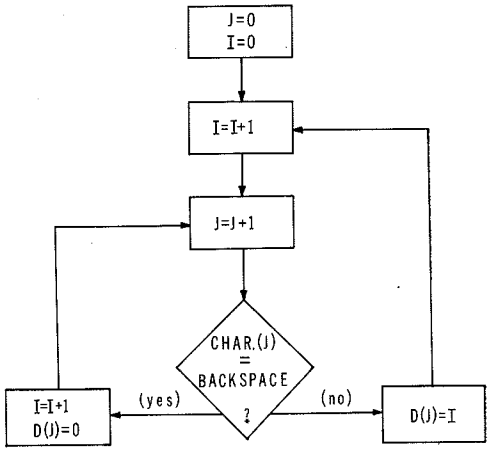
Several efficient algorithms for converting input lines to canonical form were developed. The technique currently used in the SCRIPT system is illustrated in Fig. 3.

The conversion to canonical form involves three steps.

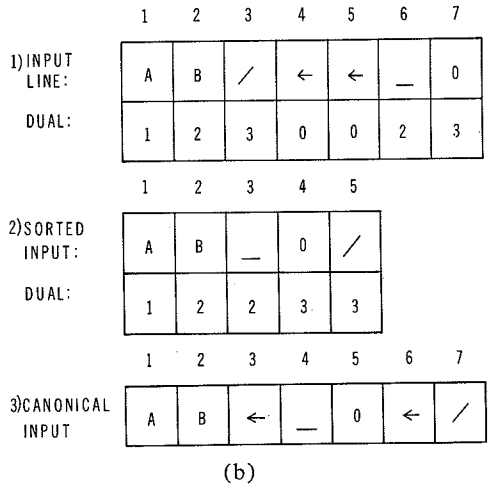
1) For each character of the input line, its "dual" is determined. The dual might be viewed as a number designating the column in which the character is logically to occur. Backspace characters are assigned a dual value of zero. The algorithm for calculation of the duals is illustrated in Fig. 3(a) and 3(b-1).

2) Treating each character and its dual as a single entry in a table, the table is sorted using the dual as the key. In the process of sorting, backspace characters (dual value of zero) are discarded and characters with the same dual value are arranged according to lexigraphical order (see Fig. 3(b-2)). This last point is made to standardize character strings, such as, "O (backspace) /" and "/ (backspace) O".

3) The canonical input line is then formed from the sorted input line by placing a backspace between characters with the same dual value, as shown in Fig. 3(b-3).



(a)



(b)

Fig. 3. (a) Algorithm to calculate dual. (b) Steps to form canonical input.

Basic Text Formatting Features

In the printer, the manuscript file is formatted and printed on either the typewriter terminal or high-speed printer. The formatting is controlled explicitly by two factors: 1) parameters of the PRINT request, and 2) format command lines embedded within the text. The rules of formatting under "fill" mode also implicitly affect the result as described above.

The PRINT request parameters designate certain global dispositions, such as output device type and/or selective output. These parameters are itemized in Table II with their abbreviations.

The PRINT format command lines provide detailed specifications on manuscript layout. The command lines may be placed anywhere in the text and take effect immediately during formatting.

Rather than supplying a limited number of "format modes," the commands are very basic. A large number of complex layouts may be built up by using a sequence of these basic formatting commands. The present commands are listed in Table III; they should be self-explanatory. They are grouped according to normal use, but may be used in combinations to produce far more general effects.

TABLE II
PRINT REQUEST PARAMETERS

default	on-line typewriter output
default	continuous form paper
STOP (ST)	single page paper
OFFLINE (OF)	off-line printer output
TRANSLATE (TR)	printer without full character set
CENTER (CE)	align to center of wide printer paper
PAGE XXX	print starting at page xxx

TABLE III
LIST OF PRINT FORMAT COMMANDS

	Page Format
.pl n	set page length to n
.bm n	set bottom margin to n
.tm n	set top margin to n
	Spacing
.ds	enter double-space mode
.ss	enter single-space mode
.sp (n)	skip n blank lines
	Paragraph Format
.ll n	set line length to n
.in n	indent all following lines n spaces
.of n	offset n spaces
	Page Control
.pa (n)	eject to new page immediate, print heading and page number
.pn mode	enter designated page numbering mode
.he heading	set page heading
	Format Mode
.br	do not "fill" between the immediately preceding and following lines
.fi	enter "fill" mode
.nf	enter "nofill" mode
.ce	center the next line
	Special Features
.rd (n)	accept next n lines from typewriter
.cp n	page eject if within n lines of bottom margin
	Manuscript Layout
.ap name	accept further input from file "name"
.im name	imbed input from file "name" and then resume original input stream

Inserts and Complex Layout Specifications

The append and imbed features were conceived as a means to connect manuscripts that had been subdivided for editing efficiency. In practice, these control features have been extremely useful for a variety of purposes. Some of the more common arrangements include:

1) incorporating "standard" paragraphs or sections into several different manuscripts;

2) separating a document into company confidential sections and nonconfidential sections, and using two or more master SCRIPT files to pull together the appropriate pieces;

3) collecting often-used complex sequences of format control words that can then be used as a "macro" by imbedding to specify complex layouts or standard paragraph forms.

Logical Topology Data Structure

Most of the operations of the printer use straightforward, conventional programming techniques. The major obstacle is processing the fill mode, complicated by the presence of underscored and overprinted characters. Furthermore, the mechanisms available for overprinting are significantly different in the typewriter terminal and in the line printer. To overcome these interrelated problems, an intermediate data structure organization is used to represent the logical topology of the line being formatted.

Each character of an input line is converted into a link element as shown in Fig. 4(a). The link elements are connected as illustrated in Fig. 4(b). Note that backspaces do not explicitly appear in the topological structure.

Only the top level of the topological structure, the "primary line" (the characters ABO in Fig. 4(b)), is involved in formatting. Sufficient text lines are read to keep the number of characters in the primary line greater than the designated output line length. An implicit blank is inserted between successive text lines.

Two passes over the primary line are needed to justify the left and right margins. During the first pass the primary line is scanned up to and including the last complete "word" (group of nonblank characters separated by blanks) contained within the length desired for justification. The number of words is determined along with the number of spaces remaining between the last word and the required line length. Dividing the number of spaces needed by one less than the number of words produces the number of extra spaces to be inserted after each word for correct justification. Unfortunately, it is not possible to insert fractional spaces. Fractional components are, therefore, accumulated until at least a half space accrues (see Table IV). When a half space accumulates, a whole space is inserted into the line and subtracted from the accumulated sum of fractions.

The second pass is required to record the added spaces. Each link element contains a multiplier field initially set to one. During the second pass the multiplier for the appropriate blank link elements is increased without altering the data structure.

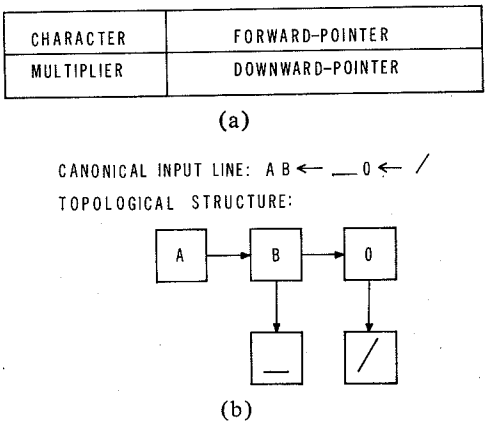


Fig. 4. (a) Link element. (b) Logical topology structure.

TABLE IV

JUSTIFICATION

(The character "." represents a blank.)

```
|<- - - - - - - - - - - - - - - - Desired Length - - - - - - - - - - - - ->|
 1   2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 A   B  .  C  D  .  E  F  .  G  H  I  .  J  K  L  .  M  .  N  O  P  R
<- - - ->  <- - - ->  <- - - ->  <- - - ->  <- - - ->  <- - - ->  <->
    1          2          3          4          5          6
```

Desired Line Length = 20
Number of Complete Words − 1 = 5
Number of Spaces Needed = 2
Number of spaces to be inserted between every word = 0
Number of Extra Spaces per Word = 2/5 or 4/10

```
|<- - - - - - - - - - - - - - - - Desired Length - - - - - - - - - - - - ->|
 1   2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 A   B  .  C  D  .  .  E  F  .  G  H  I  .  .  J  K  L  .  M
       (4/10)       (8/10)       (2/10)       (6/10)          (0)
```

The logical topology data structure is sufficiently flexible to be extended for a variety of other facilities, such as interpreting "tab" characters. Similarly, the second pass of the justification process can be altered to allow other distributions of the additional blanks, such as collecting blanks after end of sentence periods.

The logical topology data structure plays an important role in generating the correct output form. For the typewriter, the structure is "unfolded" all at once, inserting backspaces in front of the downward characters and adjusting for the multiplier. For the line printer, it is possible to print only a single line at a time. Overprinting is effected by printing multiple lines without advancing the paper. This is accomplished by "peeling" the data structure one level at a time, starting with the primary line.

III. Follow-On System

The present SCRIPT system was designed for the specific and somewhat limited objectives described in the Introduction. Modular construction has allowed expansion of the system in an organized manner. The present system is, therefore, the culmination of an adaptive process.

The evolution of the SCRIPT system has revealed limitations and shortcomings in present-day manuscript processing systems. Very recent and mostly unpublished work, such as Ken Thompson's regular expression algorithm [22], presents new concepts for on-line manuscript processing.

Many of these innovations are planned for the follow-on SCRIPT system now in its early stages (see Fig. 5). Principal concerns in the new design have been: 1) a more powerful and flexible editor command language, 2) the ability to edit a number of files at once, 3) stored editor programs, 4) floating format commands, and 5) a much more powerful pattern-matching mechanism.

The increased availability of special output devices, such as character display scopes and photocomposers, allows some interesting applications with no fundamental additional problems.

Command Language

The principal design criterion of the new command language was convenience of use. The main points were:

1) allowing several commands to be placed upon a single input line
2) shortest reasonable mnemonic names for commands
3) adaptability to user preferences
4) prevention of nuisances, particularly required upper-case characters
5) flexibility.

Most commands have the form:

address,address**command**arguments

meaning "perform the command with the arguments specified on each line between the two addresses." If only one address is given the command is performed only upon the indicated line. An address may be either

1) a line number
2) the character '.', meaning the current line
3) a function whose value is a line number
4) an address plus or minus an address.

The address function, denoted by slashes surrounding a search string, searches the text following the current line for the specified string, returning a value which is the number of the line where the first match occurred. Hence, to find the first line after the current line which contains the string 'bean', and change the string 'Boston' on that line to the string 'Cleveland'

/bean/**substitute**/Boston/Cleveland/

Buffers

Buffers are text receptacles. One might conceive of the text in each buffer as a single-dimension stream of characters or lines, and the set of buffers as multi-dimensional. Some buffer is always designated the "current buffer." The current buffer is analogous to the current file of the present SCRIPT editor. The other buffers may be considered auxiliary collections of text, which may be loaded from the current buffer, or inserted into a text stream. These two operations—"loading" and "expanding"—are the only ones permitted upon auxiliary buffers. This restriction arose from the desire to hold the number of arguments of editor commands to a minimum. Most commands, therefore, imply that the operation is to be performed upon the current buffer.

Before a manuscript is edited the text must be read into a buffer from a file. The reading operation is really more a logical than a physical operation. Although the system attempts to keep as much text as possible in core, large manuscripts can easily exhaust the available space. One solution to this problem would be to restrict the user to editing only a limited amount of text at a time. That is, the user would manually page the text to be edited into and out of buffers. We have decided to let the system automatically do this paging. When space is scarce, a buffer is attached to a disk file, and some of the text logically in the buffer is paged out onto the disk file. To hold down the time for the reading operation, only a portion of the file is actually read into the buffer. The rest is left in the file, which is logically appended to the end of the buffer.

To conserve both storage and time we employ a hierarchy of data structures. The current line is the most likely to be accessed. Each current-line character is therefore stored with forward and backward pointers to allow easy manipulation. All other text in core is maintained in strings with forward and backward pointers and a charac-

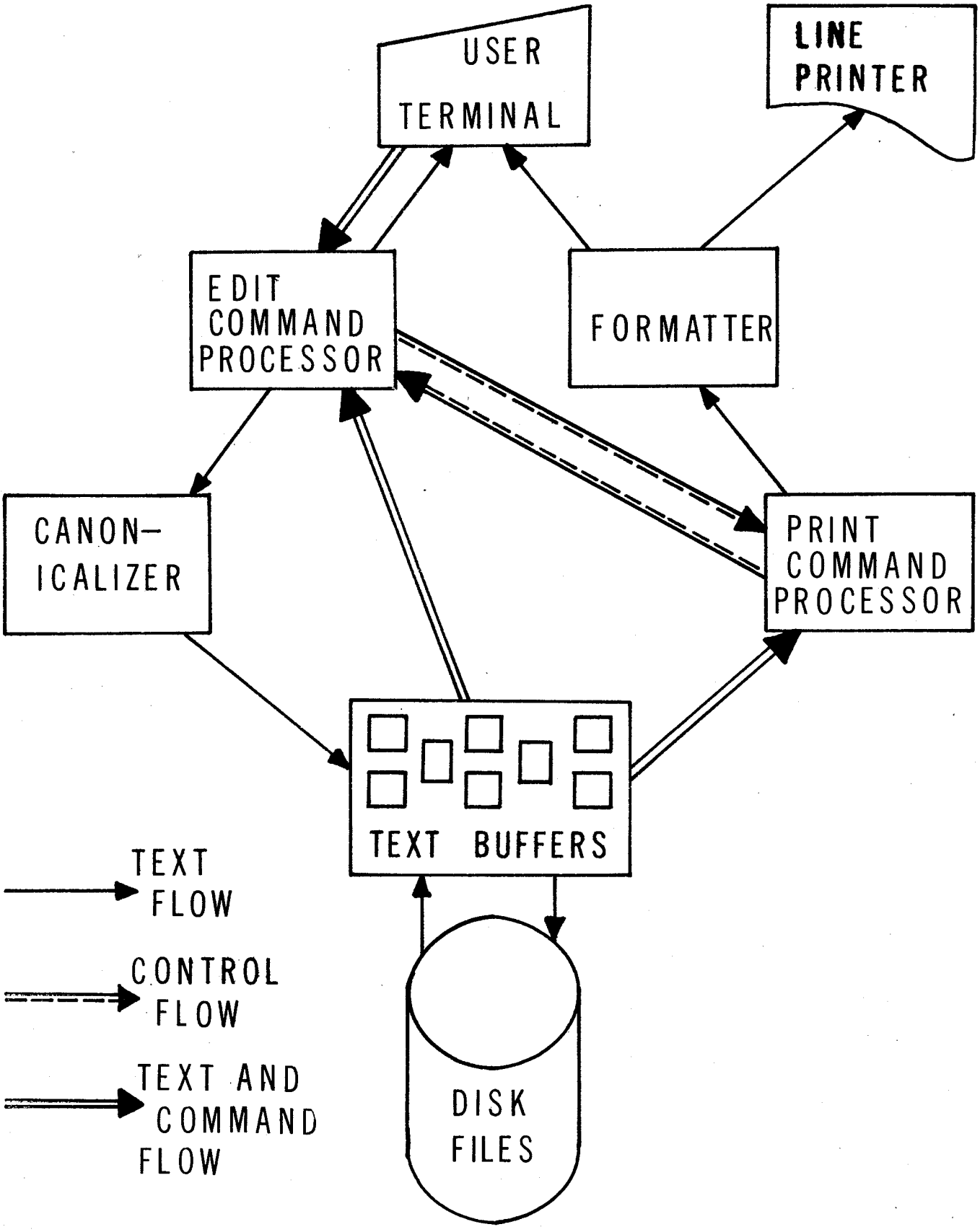


Fig. 5. System command and data flow.

ter count. Text out on disk, however, is kept in packed-character format.

Command Language Implementation

The interpreter scans the command input stream from left to right, processing each address and command as it is encountered. Addresses are calculated and pushed on top of an address stack. When a character that cannot be the beginning of an address is encountered, the interpreter compares this character and those following with entries in the command name table. The longest names in the table are matched first (this allows long names to begin with the same sequence as shorter names). When a match is found, the interpreter calls the designated command processor after removing the command name from the command stream. Each command processor examines the address stack for the addresses required for its task and removes its arguments from the command stream. After completion of a command the addresses collected before its execution are removed from the address stack. A command allows the user to redefine entries in the command table and thus to name commands in the way easiest for him to remember.

Editing Programs

Since the new SCRIPT editor is a general text manipulator, one may wish to write text manipulation functions in the SCRIPT command language. If one allows the expansion of a buffer into the command input stream, editor programs have been effected. To fully generalize this process one only needs a recursion stack. Each frame on the recursion stack indicates the previous source of the command input stream. To allow programs to be a bit fancier (and probably more efficient), labels, "go to" statements and conditionals are added. Each of these is carried out as an editor command. The "go to" command searches text in the current buffer and all predecessors on the recursion stack for the specified label definition.

Debugging editor programs often becomes a necessity. To help debug a mode may be set to cause the editor to type a message at each change in the recursion level; and the **trace** command causes the relevant information from the recursion stack to be typed out in inverse chronological order.

Editor Programs at Format Time

New format commands allow editing programs to be imbedded in the manuscript itself. These editing programs are executed when they are encountered in formatting the manuscript. This allows the contents of buffers to be changed during formatting. One intended use is placement of footnotes. When the user refers to a footnote, he uses the editor to add the footnote text to the footnote buffer. The system formats the text as it is added to the buffer and inserts this formatted text when the bottom of the page is sufficiently close, emptying the buffer afterwards.

With the arithmetic features of the editor, running footnote numbers can also be maintained. These numbers may be inserted into the text by a number-to-text conversion.

Floating Formatting Commands

Formatting commands may be interspersed freely in text. The only logical difference between floating command requirements and the current requirement that formatting commands must appear at the beginning of lines is that the floating commands may reasonably occur in the middle of a "word." The present requirement of a new line would cause a blank to be inserted, terminating the "word." The advantage to the new system comes from the greater compactness of the notation and the removal of the focus on lines as the principal subunit of the file. One result of this change of focus is the allowance of multiple-line headers. A floating formating command begins the header collection and another closes it.

Floating command coding requires setting aside a character (known as a "kludge" character) to denote a formatting command. We chose the percent sign for this purpose. Note that escape conventions [21] allow overriding the special meaning of the percent sign for use in text.

The header can be set to the string TITLE on one line and DATE on the next by the following:

%g(header)TITLE**%b**DATE%*

The formating command "group" (**%g**) interprets the characters within parentheses immediately following the **g** as a buffer name. The text following the right parenthesis up to the "terminate" (%*) command is formatted and placed in the designated buffer. The "break" command (**%b**) has the effect of a carriage return, causing TITLE to be formated on a separate line from DATE. At the top of every page the system outputs the contents of the buffer named "header."

Pattern Matching

In the present SCRIPT system only simple strings may be located or changed. It would, however, often be advantageous to have a pattern matching capability like that of SNOBOL [14], [18], [19]. Thompson's regular expression compiling algorithm [22] satisfies this need. A regular expression describing the string to be matched is compiled into machine code; the code is then executed, returning either the string found or a failure. Experience with QED [16] on the 7094 has proven this method very efficient.

The usual regular expression operators—alteration (**I**), repetition (*), and concatenation (implied)—as well as parentheses, are allowed. Operands may be either ordinary

TABLE V

SPECIAL CHARACTERS

Character	Meaning
.	matches any character
¢a	matches any alphabetic character
¢n	matches any numeric character
$	matches the character after the end of the line
@	matches the character before the beginning of the line

or special characters. Some of the special characters are shown in Table V.

Escape conventions permit referencing these characters with their usual meaning, and the extension of the character set. The string "¢c." (an escaped period) matches only a period; "¢a" (escape-a) is an extended character.

The use of regular expressions for representing patterns does not complicate the simple tasks served by string patterns—a string is still represented as a string. Regular expressions simply add the capability to specify 1) classes of characters, 2) the "or" of several characters, strings, or expressions, and 3) the indefinite repetition of a character or expression. These capabilities allow matching awkward strings without typing explicit characters on the line: for example, mispelled words ('algorythmm' would be matched by 'algo.*m'), sentences (matched by '.*¢c.'), or the first three characters on the line ('@...').

IV. CONCLUSIONS

The basic components to develop on-line manuscript processing systems have existed for some time. Recently there has been considerable commercial and academic interest in developing such systems. Unfortunately, primarily because of lack of published information, many of these projects have unintentionally "reinvented the wheel." This paper describes the design and development of a convenient and comprehensive manuscript processing system, and points out desired features for more advanced systems as guidelines for future exploration.

REFERENCES

[1] D. Murphy, "TECO," Bolt Beranek and Newman, Cambridge, Mass., Memo, November 1966.

[2] L. P. Deutsch and B. W. Lampson, "An online editor," *Commun. ACM*, vol. 10, pp. 793–803, December 1967.

[3] A. L. Samuel, "A new editing program," MIT Project MAC, Cambridge, Mass., Memo. MAC-M-167, June 1967.

[4] M. V. Mathews and J. E. Miller, "Computer editing, typesetting, and image generation," *Proc. Fall Joint Computer Conf.*, 1965.

[5] D. J. Edwards, "TECO 6," MIT Project MAC, Cambridge, Mass., Memo. MAC-M-191, October 1964.

[6] M. M. Zryl and A. P. Mullery, "Text editing on the APL/360 terminal system," IBM Watson Research Laboratories, Memo., July 1967.

[7] J. H. Saltzer, "TYPSET and RUNOFF, memorandum editor, and type-out commands," MIT Project MAC, Cambridge, Mass., Memo. MAC-M-193-2, January 1965.

[8] R. C. Daley, "ED, a context editor for card image files," MIT Project MAC, Cambridge, Mass., Memo. MAC-M-195, November 1964.

[9] IBM Corporation, "Introduction to DATATEXT," Form J20–0011.

[10] IBM Corporation, "TEXT/90," File 7090–ZO IBM 0022ZO.

[11] IBM Corporation, "Administrative terminal system (ATS)," Form H20–0185–1.

[12] P. Sampson, "PDP–6 TECO, July 1965 version," MIT Project MAC, Cambridge, Mass., Memo. MAC-M-250, July 1965.

[13] J. J. Corbato *et al., The Compatible Time-Sharing System, A Programmer's Guide* (Memo. Ditto, Modify Commands), Cambridge, Mass., MIT Press 1963.

[14] S. E. Madnick, "String processing techniques," *Commun. ACM*, vol. 10, pp. 420–424, July 1967.

[15] IBM Corporation, "Magnetic Tape Selectric Typewriter" Training Guide, Form 543–0503–1.

[16] K. Thompson, "QED as of 8/5/67," Bell Telephone Laboratories, Murray Hill, N.J., Memo. B0088, August 1967.

[17] IBM Cambridge Scientific Center, "CP/CMS user's guide," Cambridge, Mass., Rept. 320–2015, October 1967.

[18] S. E. Madnick, "SPL/1: A string processing language," IBM Cambridge Scientific Center, Cambridge, Mass., Rept. 36.006, June 1966.

[19] D. J. Farber, R. E. Griswold, and I. P. Polansky, "SNOBOL, a string manipulation language," *J. ACM*, vol. 11, January 1964.

[20] W. D. Mathews, "EDIT, An online string manipulator," MIT, Cambridge, Mass., Project TIP Memo., March 1968.

[21] "Escape conventions," *Multics System Programmer's Manual*, Section BC2.04, MIT Project MAC, Cambridge, Mass.

[22] K. Thompson, "Regular expression search algorithm," *Commun. ACM*, vol. 11, pp. 419–422, June 1968.