pdp11

RSTS/E

Text Editor Manual

Order No. DEC-11-UTEMA-A-D

digital

RSTS/E

Text Editor Manual

Order No. DEC-11-UTEMA-A-D

digital equipment corporation · maynard, massachusetts

First Printing, May 1976

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11

CONTENTS

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

This manual describes the features and operation of EDIT, a BASIC-PLUS program distributed with the RSTS/E standard system library. The manual is written for the RSTS/E user with little or no knowledge of character editors.

For more information on RSTS/E guides and manuals, consult the RSTS/E Documentation Directory.

For a quick reference to a subject in this guide, use the following list.

If you need to know about	**See Section**
Loading EDIT	1.3.1
Creating a new file	1.3.3
Dividing text into pages	2.1.4
Editing multiple-page files	2.1.5
Locating the text to be edited	2.2
Moving the location of the reference pointer	2.3
Deleting text stored in the buffer	3.1, 3.2
Inserting text in the buffer	3.3
Changing text in the buffer	3.4, 3.5
Terminating EDIT	4.1

CHAPTER 1

INTRODUCTION

1.1 EDIT OVERVIEW

EDIT is a character-oriented text editing program written in BASIC-PLUS for use under the RSTS/E operating system. Text editing involves two programs: 1) EDIT, which manages input and output; and 2) EDITCH, which processes commands. For clarity, however, this manual refers to only one program, EDIT. EDIT, which is operated by use of commands typed at the terminal, reads ASCII files from any input device, makes specified changes, and writes on any output device. The basic editing process can be divided into three sequential steps:

1. Reading of input text into an internal buffer
2. Changing the text stored in the buffer
3. Outputting the revised text to a new file

These steps are described in Chapters 2, 3, and 4. The following sections list EDIT terms and definitions and describe the process of running EDIT.

NOTE

The EDIT program conforms to the standards of the DOS/BATCH Text Editor program, EDIT, except where noted in the description.

1.2 EDIT TERMS AND DEFINITIONS

The EDIT system program enables a user to perform editing operations on ASCII text. (To edit other types of data, the ODT system program can be used.) The program stores the text to be edited in an intermediate area of memory called a buffer. It does not alter text in the input files.

To begin editing operations (after the input and output specifications have been typed), the user must transfer text from the input file(s) to the buffer. After text in the buffer is edited, the user must transfer the result to the output file(s). During the editing session, the user can store text in a special location called a save buffer, which permits him to transfer text from the save buffer to any location or locations in the main buffer. This procedure simplifies moving text from place to place and reduces the amount of typing required to insert similar material in separate places in the text. Additionally, text in the save buffer can be treated as an EDIT command line. This capability serves as a macro function whereby the user can run a series of EDIT commands stored in the save buffer by typing only one other EDIT command. Figure 1-1 is an overview of editing operations.

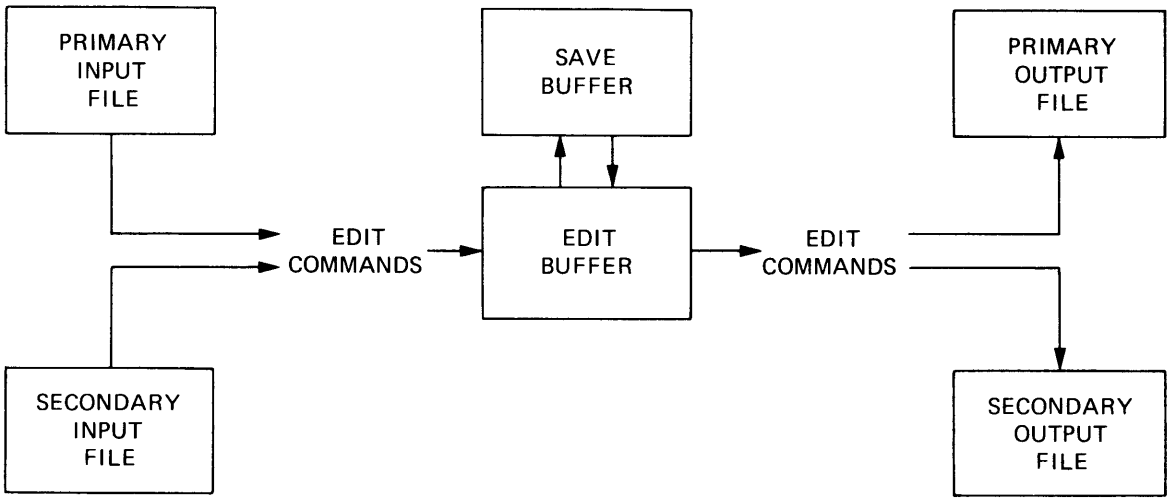


Figure 1-1 Editing Operation Overview

To facilitate editing operations, EDIT recognizes different units of text. A form feed (FF) character terminates a page of text. A page is the most convenient unit of text to transfer into and out of the buffer. A carriage return (CR) and line feed (LF) character sequence; a LF, CR, and NUL character sequence; or the escape (ESC or ALTMODE) character terminates a line[1] of text. A character is the smallest unit of text that can be operated on by EDIT.

EDIT refers to text in the buffer by using a character location pointer called Dot. Dot is considered to reside between any two characters. At the start of editing operations, Dot precedes the first character in the buffer. Dot is moved during editing operations according to the type of editing operation being performed. The user can refer to text in the buffer as so many characters or lines preceding or following Dot.

Additionally, a temporary reference pointer called Mark can be used to refer to text. Mark remembers a location by moving to Dot and conditionally remaining there while Dot moves on to some other place in the text. The user can refer to an amount of text between Dot and Mark. This type of reference does not require the user to specify the number of characters or lines involved in the editing operation.

To edit text, the user specifies a command or series of commands in response to the asterisk printed by EDIT. The commands are classed according to the type of operation they perform. Table 1-1 describes the classes of EDIT commands.

Table 1-1
Classes of EDIT Commands

Class	Description
Input and Output	Transfers text into and out of the buffer. Also prints the contents of the buffer at the terminal.
Dot Manipulation	Moves the character location pointer, Dot, without altering text within the buffer. Also establishes the temporary pointer, Mark.
Character Search	Finds a specified occurrence of text within the buffer to facilitate editing. Additionally, positions the pointer in relation to the specified text.
Character Manipulation	Adds to and removes from the buffer characters or lines of text.
Miscellaneous	Performs related editing operations such as opening and closing files, transferring text to and from the save buffer, and executing predefined series of EDIT commands.

Some EDIT commands accept an argument that specifies either the particular portion of text to be affected or the number of times to perform the command. The interpretation of the argument depends on the type of command. Table 1-2 shows the possible arguments and their meanings.

Many EDIT commands are character-oriented. That is, they affect a specified number of characters preceding or following Dot. The argument of such commands specifies the number of characters in the buffer on which to operate. The number of characters specified by the argument n is the same forward as backward (-n). LF, CR, and NUL characters, although not printed, are embedded in text lines, counted as characters in character-oriented commands, and treated as any other text characters.

Some EDIT commands are line-oriented. The argument of such commands specifies the number of lines on which to operate. Because EDIT counts the line-terminating characters to determine the number of lines on which to operate, an argument n does not affect the same number of lines forward (positive) as it affects backward (-n). For example, the argument -1 applies to the line beginning with the first character following the second previous end-of-line and ending with the character preceding Dot. The argument 1 in a line-oriented command, however applies to the text beginning with the first character following Dot and ending at the first end-of-line. Thus, if Dot is at the center of the line, the argument -1 affects one and one-half lines backwards from Dot and the argument 1 affects one-half line beyond Dot.

Character search and manipulation commands operate in either of two modes: Command mode or Text mode. The modes are merely different ways of specifying a text object following a command. The text

[1] This line is the same unit used by BASIC-PLUS and defined in Section 5.3 of the BASIC-PLUS Language Manual.

Table 1-2
Command Arguments

Argument	Description
n	A decimal number between 1 and 32767; it is assumed to be positive unless preceded by a minus (-) character. The absence of n implies a 1 (or -1 if a - character precedes a command). n can be the number of characters or lines forward or backward (-) or the number of times to execute the operation.
0	Indicates the text between the beginning of the current line and Dot.
@	Refers to the text between Mark and Dot.
/	Refers to the text between Dot and the end of the text in the buffer.

object can be a single character, a group of characters (called a string), or several lines of characters. In no case, however, can a single line of a text object exceed 255 (decimal) characters.

If the text does not contain CR or LF characters and is small enough to fit on a single typed line, Command mode can be used to specify the text object. In Command mode, EDIT expects the first character following a search or manipulation command to be a delimiting character for the desired text object. EDIT uses as the text object the characters between the delimiter and the next occurrence of the delimiter. The delimiting character, therefore, cannot appear within the text object. Additionally, neither a CR nor a LF character can appear in the text object.

To specify the word INPUT as the text object in Command mode, the following string can be used with the search or manipulation commands.

```
/INPUT/
```

EDIT uses as the text object the characters INPUT between the / delimiters.

In Command mode, any printable ASCII character that does not appear in the text object can be used as a delimiter. For example, if the string /INPUT/ is the text object, the character A could be used as a delimiter, as shown in the following string:

```
A/INPUT/A
```

EDIT uses the A as the delimiter and treats the intervening characters /INPUT/ as the text object. If the delimiting character appears in the text object, EDIT attempts to interpret the remaining characters as commands.

If the text object contains CR and LF characters or is too long to fit on a single typed line, Text mode must be used to specify the text object. The user causes EDIT to enter Text mode by pressing the RETURN key following the command requiring a text object. EDIT enters Text mode by generating a carriage return/line feed. This action allows the text object to be typed at the start of a new line.

In Text mode, EDIT accepts as part of the text object each line typed. Pressing the RETURN key in Text mode causes EDIT to generate a carriage return/line feed sequence and to enter a CR and a LF character as part of the text object. To terminate the text object, simply press the LINE FEED key. The LF character is not entered as part of the text object. EDIT prints the asterisk prompting character again.

NOTE

To insert a LF character as part of the text object, the LINE FEED key must be pressed as the first character of the text object. EDIT generates a carriage return/line feed sequence but additionally enters the LF character in the buffer. The second time the LF key is pressed, EDIT terminates Text mode normally and does not enter the second LF character as part of the text object.

1.3 RUNNING EDIT

1.3.1 Loading EDIT

To load EDIT, type the following command while at BASIC-PLUS command level:

```
RUN $EDIT
```

EDIT runs and prints its version number and the number sign (#) character to indicate that it is ready to accept input and output specifications. This information should be entered in the following format:

#OUT1.EXT,OUT2.EXT<IN1.EXT,IN2.EXT/B

IN1.EXT and OUT1.EXT are the primary input and output files, respectively; IN2.EXT and OUT2.EXT are the secondary input and output files. The input and output files can be denoted by any valid RSTS file specification, including a device, filename, extension, project-programmer number, and protection code. If no extension exists in an input file specification, EDIT assumes an extension of BAS. If an output file is not specified, EDIT creates the input file, and destroys any existing file with the same name. If no extension is given to an output file, EDIT creates the output file with a BAS extension.

The /B option in the file specification conditions the EDIT program to treat a line feed (LF) character in the text as a BASIC-PLUS line continuation character.[1] The program treats a LF character in the text, therefore, as a line feed/carriage return without requiring the presence of the carriage return (CR) character in the file. For files with an explicit BAS extension or with an assumed BAS extension, EDIT automatically enables the /B option. Specify the /B option when EDIT is required to handle BASIC-PLUS continuation lines in a file with an extension other than BAS.

1.3.2 Temporary Backup Files

If the primary input and output files have the same filename and extension, EDIT creates a backup file by renaming the primary input file after editing is complete. During editing operations, the program uses a temporary filename of EDITnn.TMP for the primary output file. The designation nn is the job number under which EDIT runs. Upon termination of the editing session, the program gives the primary input file an extension of BAK to signify the backup version of the file. EDIT renames the file EDITnn.TMP with the specified filename and extension to signify the new, edited version. If the primary input and output have the same filename and extension (implicit or explicit), EDIT creates the backup version of the file and renames the edited version with the specified filename and an extension of BAK.

If EDIT encounters any errors while accessing the specified files, it prints the related BASIC-PLUS error text in the following format:

error text - ERROR

EDIT then prints the number sign (#) character, after which the user can type another response.[2]

If no errors are encountered, EDIT chains to the EDITCH program, which prints a blank line followed by the asterisk (*) character. The asterisk is a prompting indicator that signifies EDIT is ready to accept commands from the user. The following sample dialogue shows the procedure.

```
#FILE.BAS<FILE.BAS/B

  *
```

EDIT accesses FILE.BAS under the current user's account, opens it for input, opens the file EDITnn.TMP for output, and chains to EDITCH, which prints the prompting indicator. An output file must be specified when attempting to edit an existing file. If an output file is not specified, EDIT creates the input file and destroys any existing file with the same name.

1.3.3 Creating A File

If the purpose of an editing session is to create a file for which there is no input file, the user need not specify a primary input device. When the text has been inserted, edit the text or close the file by specifying appropriate commands. The format for creating and closing a file is:

```
RUN $EDIT
(version number of EDIT)
#dev:filename.ext<prot>

*I <CR>

text <CR>

<LF>
*EX <CR>

#^Z

READY
```

[1] This action differs from the DOS/BATCH EDIT program. See Section 2.3.2 of the BASIC-PLUS Language Manual for a description of continuation lines in BASIC-PLUS statements.

[2] BASIC-PLUS error messages are summarized in Appendix C of both the BASIC-PLUS Language Manual and the RSTS-11 System User's Guide.

EDIT prints the number sign (#) character and waits for the user to specify 1) a device on which the file will be created and 2) the name, extension, and protection code of the file. If no device is specified, EDIT automatically creates the file on the system disk. The program automatically assigns a BAS extension if no extension is given. EDIT opens the file for output and thus destroys any currently existing file with the same name if the protection code permits. The primary input file is the user's keyboard.

If EDIT encounters no errors, it prints a blank line followed by an asterisk (*) character indicating its readiness to accept a command. Type the I command, indicating text is to be inserted, and the carriage return (CR) key. Lines of text may then be inserted, each terminated by the CR key, which enters a carriage return (CR) and a line feed (LF) character in the buffer. Pressing the line feed (LF) key terminates the I command but does not enter a LF character in the buffer. EDIT responds with an asterisk (*) character indicating command level. Type the Exit (EX) command to close the editing session. The CTRL/Z combination terminates EDIT and returns control to the system monitor.

1.3.4 Opening a Currently Existing File for Editing

Reopen a file for editing, if necessary, by either of the following methods:

Example 1:

```
RUN $EDIT
(version number of EDIT)
 # filename.ext<prot> <filename.ext

*R
*/L
```

or

Example 2:

```
EDIT filename.ext

*/L
```

If the file was created with an extension other than BAS, the extension must be specified after the filename.

The R command in Example 1 instructs EDIT to read the first page of the specified file into the buffer. EDIT positions Dot at the beginning of the buffer. EDIT then returns to Command mode by printing an asterisk (*) character to which the user can respond with a desired EDIT command. The /L command prints the contents of the buffer on the user's terminal.

In Example 2, when the user types the filename and extension on the same line as the CCL command EDIT[1], EDIT opens for input the file in the current account, creates a backup file, and automatically reads the first page into the buffer. The R command need not be typed. EDIT prints an asterisk (*) character indicating its readiness to accept EDIT commands. By typing the /L command, the entire contents of the buffer are listed on the terminal.

Typing the EX command ends the editing session. In Example 1, control returns to EDIT, which prints the #character. In Example 2, typing EX returns control to the system monitor, which prints READY.

[1] This feature is optional and may not be available on all RSTS/E systems. If it is not available on the system, the monitor prints the WHAT? error message.

CHAPTER 2
READING INPUT TEXT INTO INTERNAL BUFFER

2.1 TRANSFERRING TEXT TO BUFFER

This section describes the commands and procedures required to read text from the input files to the buffer, list the contents of the buffer on the terminal, verify the location of Dot, divide text into pages, and edit multiple-page files. See Appendix A for some examples of the uses of EDIT, Appendix B for a summary of EDIT commands, and Appendix C for an explanation of EDIT error messages.

2.1.1 Read and Edit Read Commands

Before editing can be performed on text, the input file must be read into the buffer. The Read (R) and Edit Read (ER) commands provide two ways of transferring text to the buffer for editing. The command R reads text from the primary input file; the command ER reads text from the secondary input file. For example, when the user loads EDIT by way of a RUN $EDIT command and specifies the input and output files, the R command reads the first page of the primary input file into the buffer and editing operations can begin. EDIT transfers text to the buffer until one of the following conditions occurs:

1. A FF character is encountered.
2. The buffer is 500 characters from being full.
3. An end-of-file is encountered.

EDIT reads text from the input files and appends it to the current contents of the buffer. There are no arguments with the Read and Edit Read commands. Each command thus reads a page of text at a time from the appropriate input file. Following execution of an R or an ER command, the program positions Dot and Mark at the beginning of the buffer.

The following is an example of creating a file using the R and ER commands. Each comment refers to the appropriately marked lines in the example:

```
#PAUL<ELEANOR, KATHY

①*R/L
THIS IS PAGE ONE
OF THE PRIMARY
INPUT FILE.

②*R/L
THIS IS PAGE ONE
OF THE PRIMARY
INPUT FILE.


THIS IS PAGE TWO
OF THE PRIMARY
INPUT FILE.
③*ER/L
THIS IS PAGE ONE
OF THE PRIMARY
INPUT FILE.
```

① Reads the first page of the primary input file, ELEANOR.BAS, into the buffer. Dot is placed at the beginning of the buffer. The contents of the buffer are listed on the terminal beginning at Dot and ending with the last character in the buffer.

② Reads the second page into the buffer and appends it to the current contents. The contents of the buffer are listed on the terminal beginning at Dot and ending with the last character in the buffer.

③ The first page of the secondary input file, KATHY.BAS, is read into the buffer and is appended to the current contents of the buffer. The contents of the buffer are listed on the terminal beginning at Dot and ending with the last character in the buffer.

(continued on next page)

(continued from previous page)

```
THIS IS PAGE TWO
OF THE PRIMARY
INPUT FILE.
THIS IS PAGE 1
OF THE SECONDARY
INPUT FILE.

④*EX

#
```

④ The editing session is ended and the contents of the buffer are written to the primary output file. The result is a file named PAUL.BAS containing the contents of the buffer and any remaining text in the primary input file.

2.1.2 Listing Lines of Text

The List (L) command prints at the terminal lines of text as they appear in the buffer. An argument preceding the L command indicates the portion of text to print. For example, the command, 2L, prints on the user's terminal the text beginning at Dot and ending with the second end-of-line character. Neither Dot nor mark is altered by the L command. Arguments and their effect upon the List command are described as follows:

- nL Prints at the terminal n lines beginning at Dot and ending with the nth end-of-line character.
- -nL Prints at the terminal n lines beginning at the nth end-of-line character preceding Dot.
- 0L Prints the current line up to Dot.
- @L Prints the text between Dot and Mark.
- /L Prints the text between Dot and the end of the buffer.

In the following example, EDIT is loaded, an input and an output file are specified, the first page of the input file is read into the buffer, and the first line of the file is listed.

```
RUN $EDIT
EDIT V05B-11

#FBK<FBK

*R
*L
10     PRINT "R", "SIN(R)", "COS(R)"
*
```

2.1.3 Verifying Location of Dot

The Verify (V) command prints at the terminal the entire line in which Dot is located. It provides a ready means of determining the location of Dot after a search is completed and before any editing commands are given. (The V command combines the two commands 0LL.) Also, V can be typed after an editing command to allow proofreading of the results. No arguments can be specified with the V command. The locations of Dot and Mark are not changed.

2.1.4 Dividing Text Into Pages

The Form Feed (F) command merely inserts a FF character immediately after the current location of Dot in the buffer. Dot and Mark are not altered. No arguments can be specified with the F command. The F command can be used to organize text in the buffer into pages to make later editing operations easier. For example, the following series of commands sets up a file comprised of two pages:

```
#FORM

*I
THIS IS AN EXAMPLE OF CREATING A FILE ON SEVERAL PAGES.

*F
*I
THIS SENTENCE IS BEING CREATED ON A SEPARATE PAGE.

*EX

#
```

This series of commands creates a file named FORM.BAS. The text is inserted, as indicated by the I command, and the line feed (LF) character returns EDIT to Command mode. The user inserts a form feed by typing the F command. (The form feed is printed as four consecutive blank lines.) Text is inserted following the form feed in the buffer and the editing session is ended. The result is a two-page file.

2.1.5 Editing Multiple-Page Files

The Next (N) command writes the contents of the buffer to the primary output file, deletes the buffer, and reads the next page of the primary input file into the buffer. Dot and Mark are positioned at the beginning of the buffer. If the argument n is specified with the N command, the sequence is executed n times.

If EDIT encounters the end of the primary input file when trying to execute an N command, it prints N? to indicate that no further text remains in the primary input file. The contents of the buffer are transferred to the primary output file and the buffer is cleared.

The N command alone is a quick method of writing edited text to the primary output file and setting up the next page of text in the buffer. Also, use of the N command with an argument is a convenient means of quickly setting up text in the buffer, provided its page location in the primary input file is known. The N command operates in a forward direction only; therefore, negative arguments cannot be specified preceding an N command.

In the following example, an N command copies a primary input file with more than one page of text to the primary output file. Each comment refers to the appropriately marked lines in the example.

```
#FUB<FUB

① *N/L
THIS IS PAGE ONE OF A TWO-PAGE FILE.

② *N/L
THIS IS PAGE TWO OF A TWO-PAGE FILE.
③ *N/L

N?
④ *EX

#
```

① Reads the first page of the primary input file, FUB.BAS, into the buffer and lists the entire page on the terminal.

② Transfers the contents of the buffer to the primary output file, clears the buffer, and reads the next page into the buffer.

③ Transfers the current contents of the buffer to the output file, clears the buffer, and encounters the end of the file. Because the N sequence cannot be completed, EDIT prints the related N command followed by a ? on the terminal. The buffer is empty and the entire primary input file is appended to the primary output file.

④ EDIT returns to Command mode as indicated by the *. Typing the EX command ends the editing session.

2.2 CHARACTER SEARCH COMMANDS

Certain EDIT commands search for text in the buffer. These commands require a text object.

2.2.1 Get Command

The Get (nG) command is the basic search command in EDIT. The command searches for the nth occurrence of the specified text object starting at the current location of Dot. If the modifier n is not given, EDIT searches for the first occurrence of the text object. The search terminates when EDIT either finds the nth occurrence or encounters the end of the buffer. If the search is successful, EDIT positions Dot to follow the last character of the text object. EDIT notifies the user of an unsuccessful search by reprinting the related G command followed by a question mark (?) character. In this instance, EDIT positions Dot after the last character in the buffer.

Examples of arguments used with the G command follow:

nG/text/ Searches the current buffer beginning at Dot for the nth occurrence of the specified text object (text). If the search is successful, Dot is placed immediately after the text object. If the search is unsuccessful, Dot is placed at the end of the buffer. If the modifier n is not specified, EDIT searches for the first occurrence of the text object.

If the specified text object contains no carriage return or line feed characters, it can be set off from the nG command by delimiters. For example, assume Dot is at the beginning of the buffer shown below.

```
10   READ A$,B$,C$,D$,E$
```

The command 2G/$/ searches for the second occurrence of the character $ following Dot. The slash (/) character is used as the delimiter because it does not occur in the specified text object.

```
10   READ A$,B$,C$,D$,E$
                ↑
     Dot after 2G/$/
```

If the text object is long or contains a CR and LF sequence, the G command can be used in Text mode. For example, assume Dot is at the beginning of the buffer shown below:

```
 5   DIM A 25$
10   PRINT INPUT LINE 25%
15   T$ = 5NC 25% (A25%)
20   CHANGE A25% TO A25%
25   A25% (J25%) = A25% (X25%)
```

If the user desires EDIT to position Dot at line 25 and to ignore all other occurrences of 25, the following command can be used:

```
*G<CR>
 <CR>
 25<LF>
 *
```

The G command is opened in Text mode. The search string is a carriage return-line feed followed by 25. Pressing the line feed key following the text object, 25, causes EDIT to terminate the search string and to search for the occurrence of the string, 25, following an end-of-line (carriage return-line feed). EDIT positions Dot following the line number, 25.

2.2.2 Searching Primary Input File

Two commands, the Whole command and the Position command, perform a search through the whole primary input file for the nth occurrence of the specified text object. The difference between them is that the Whole command transfers the searched text to the primary output file, whereas the Position command deletes the contents of the searched buffer.

2.2.3 Whole Command

The Whole (nH) command reads each page of the primary input file into the buffer until the nth occurrence of the specified text object is found. EDIT begins at Dot and searches the current buffer until the nth occurrence of the text object is found or the end of the buffer is reached. If EDIT finds the text object, it places Dot immediately following it. If the nth occurrence is not found in the current buffer, EDIT writes the buffer to the primary output file, clears the buffer, reads the next page of the primary input file into the buffer, and continues the search. The search is unsuccessful when the nth occurrence is not found and the end of the primary input file is reached. EDIT indicates an unsuccessful search by reprinting the related H command followed by a question mark (?) character. Upon an unsuccessful search, EDIT copies the entire contents of the primary input file to the primary output file and positions Dot at the beginning of an empty buffer. The Whole command operates only in a forward direction; therefore, a minus sign (-) character cannot be specified preceding the argument n.

The user can employ an H command to copy all remaining text from the primary input file to the primary output file by specifying a nonexistent text object. The H command performs the same function as the EX command except that the H command does not end the editing session.

The following is an example of performing a search through the primary input file for the first occurrence of the character, Q. Each comment refers to the appropriately marked lines in the example.

```
#STEVE<STEVE,ABC

①*R
②*/L
  FORMAT  A, B, Q, C, D
③*H/Q/
④*0L
  FORMAT  A, B, Q*EX

#
```

① Reads the first page of the primary input file, STEVE.BAS, into the buffer. (The R command need not be specified prior to an H command unless the user desires the contents of the buffer listed prior to the search.)

② Lists the contents of the buffer on the terminal.

③ Performs a search through the whole primary input file for the first occurrence of the character, Q. A successful search is indicated by the asterisk character. EDIT places Dot following the searched character.

④ Lists the line up to Dot. The EX command transfers the contents of the buffer and the remainder of the primary input file to the primary output file and closes the editing session.

2.2.4 Position Command

The Position (nP) command is identical to the Whole (H) command with one exception. The P command deletes the contents of the buffer after it is searched, whereas the H command transfers the contents of the buffer to the primary output file. The nP command searches each page of the primary input file for the nth occurrence of the text object starting at Dot and ending with the last character in the buffer. If EDIT finds the nth occurrence, it positions Dot following the text object and positions Mark at the beginning of the page containing the searched object. If the search is successful, EDIT deletes all pages preceding the one containing the text object and positions the page containing the text object in the buffer. If the user then ends the editing session, EDIT copies the page containing the text object and all subsequent pages to the output file. If the search is unsuccessful, EDIT clears the buffer and no text is transferred to the output file. EDIT notifies the user of an unsuccessful search by reprinting the related P command followed by a ? character.

If the purpose of the editing session is to create a new file out of the second half of the primary input file, the P search saves time. The following shows the procedure for creating a file from the second half of the primary input file. Each comment refers to the appropriately marked lines in the example.

```
#BBB<ZIP,AAA

①*P/3/
②*0L
  PAGE  3*EX

#
```

① Searches the primary input file, ZIP.BAS, for the first occurrence of the text object, 3. EDIT positions Dot after the text object.

② Lists on the terminal the current line up to Dot. The EX command transfers the page containing the text object and any subsequent pages to the output file, BBB.BAS, and ends the editing session.

2.2.5 Searching Secondary Input File

A search through the whole secondary input file can be made by specifying one of two commands: 1) the EDIT Whole command transfers searched buffers to the primary output file; and 2) the Edit Position command deletes the contents of the buffer after an unsuccessful search.

2.2.6 Edit Whole Command

The nEH command performs a search through the secondary input file for the nth occurrence of the specified text object starting at Dot in the buffer. If the nth occurrence is not found in the current buffer, EDIT writes the buffer to the primary output file, clears the buffer, reads the next page from the secondary input file to the

buffer, and continues the search. The search is terminated when either the nth occurrence of the text object is found or the end of the secondary input file is reached.

If the modifier n is not specified, EDIT searches for the first occurrence of the text object. If the search is successful, EDIT positions Dot to follow the last character of the text object. The previously searched buffers are transferred to the primary output file and the page containing the text object remains in the buffer. Upon an unsuccessful search, EDIT reprints the related nEH command followed by a question mark (?) character on the user's terminal. In this instance, EDIT writes the whole secondary input file to the primary output file and places Dot at the beginning of an empty buffer.

The following set of commands performs a search for the third occurrence of the dollar sign ($) character in the secondary input file, SHEILA.BAS. Each comment refers to the appropriately marked lines in the example.

```
#ALLAN<STEVIE, SHEILA

(1)*3EH/$/
(2)*0L
  10      READ A$, B$, C$*EX

#
```

(1) Searches the secondary input file, SHEILA.BAS, for the third occurrence of the dollar sign ($) character. EDIT transfers the searched pages of SHEILA.BAS, which do not contain the third occurrence of the $ character, to ALLAN.BAS and positions Dot after the searched character in the buffer.

(2) Lists the line containing the searched object up to the location of Dot. The EX command transfers the buffer to ALLAN.BAS, clears the buffer, transfers the file STEVIE.BAS to ALLAN.BAS, and ends the editing session.

EDIT automatically appends the primary input file to the secondary input file and transfers it to the primary output file unless the contents are read into the buffer and killed. For example:

```
#ALLAN<STEVIE, SHEILA

R/L
*THIS IS THE PRIMARY INPUT FILE.
*/K
*3EH/$/
*0L
10      READ A$, B$, C$*EX

#
```

2.2.7 Edit Position Command

The nEP command is identical to the Position command except that the file searched is the secondary input file. The nEP command searches the secondary input file for the nth occurrence of the text object. EDIT deletes the searched pages of the secondary input file, which do not contain the nth occurrence of the text object, and positions the page containing the text object in the buffer. Dot is positioned after the text object. If the argument n is not specified, EDIT performs a search for the first occurrence of the text object.

The EP command can be used to create a file by 1) using the primary input file and appending text in the secondary file to it, 2) appending the primary input file to portions of the secondary input file, or 3) using portions of the secondary input file only. Descriptions of each method follow.

Example 1:
This example uses the primary input file and a portion of the secondary input file to create a file. Each comment refers to the appropriately marked lines in the example.

```
#ZUP<AAA,ABC

①*R
②*EP/THREE/
③*EX

#
```

① Reads the first page of the primary input file, AAA.BAS, into the buffer.

② Searches each page of the secondary input file for the first occurrence of the text object, THREE. The current contents of the buffer are not searched. Upon a successful search, EDIT positions the page containing the text object in the buffer and positions Dot after the text object. All previously searched pages are deleted and the unsearched pages of the secondary input file are not transferred to the buffer.

③ Transfers text in the buffer to the primary output file, transfers the remainder of AAA.BAS to ZUP.BAS and ends the editing session.

The result is a file named ZUP.BAS, which contains the first page of the primary input file, AAA.BAS. Appended to the first page of the primary input file is the page of the secondary input file, ABC.BAS, containing the text object and the remainder of AAA.BAS. For example, the primary output file would contain:

```
#ZUP<ZUP

*R/L
PAGE 1 - PRIMARY INPUT FILE.



*N/L
PAGE THREE - SECONDARY INPUT FILE.
PAGE 2 - PRIMARY INPUT FILE.



*N/L
PAGE 3 - PRIMARY INPUT FILE.
*EX

#
```

Example 2:
The following example creates a file by using a portion of the secondary input file and appending the primary input file to it. Each comment refers to the appropriately marked lines in the example.

```
#ZUP<AAA,ABC

①*EP/THREE/
②*EX

#
```

① Reads the first page of the secondary input file, ABC.BAS, into the buffer and searches for the first occurrence of the text object, THREE. If the search is unsuccessful, EDIT clears the buffer and reads the next page of the secondary input file into the buffer. This sequence is repeated until the end of the file is reached or the searched object is found. Upon a successful search, the page containing the text object is in the buffer and Dot is positioned after the text object.

② Transfers the contents of the buffer to ZUP.BAS, appends AAA.BAS, and ends the editing session.

The result is a file named ZUP.BAS, comprised of the page of the secondary input file containing the text object and appended to it the primary input file. The EX command automatically transfers the primary input file to the primary output file. An R command need not be specified.

Example 3:
The following sequence of commands uses a portion of the secondary input file to create a file.

```
#ZUP<AAA,ABC

*R/K
*EP/THREE/
*EX

#
```

The format is the same as Example 2 except that the R/K command reads the primary input file into the buffer and kills it prior to the search for the first occurrence of the text object in the secondary input file. If the primary input file contains several pages, the R/K command can be repeated as many times as necessary to kill each page of the file. The EP command searches the secondary input file for the first occurrence of the text object, THREE. When EDIT finds the text object, it positions the page containing the text object in the buffer and positions Dot after the text object. The result is a file named ZUP.BAS, which contains only the page of the secondary input file containing the text object.

2.3 DOT MANIPULATION COMMANDS

Dot is the character location pointer used by EDIT to refer to text in the buffer. Several commands are available to allow a user to manipulate the location of Dot without altering the text.

2.3.1 Setting Dot to Beginning of Buffer

The Beginning (B) command sets Dot to precede the first character in the buffer. The command allows the user to establish the initial condition of EDIT (i.e., with Dot at the beginning of the buffer) as many times as necessary. When text is initially read into the buffer, Dot resides at the beginning of the buffer. During editing, Dot is moved to different locations in the buffer. The B command can be used, therefore, to reset Dot at the beginning of the buffer. For example, to list the contents of the buffer on the terminal after editing operations have been performed, type the B command followed by a /L command. The B command can also be used to reset Dot upon an unsuccessful search for a text object. The Beginning (B) command does not use an argument and does not alter Mark.

2.3.2 Moving Dot by Character

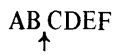
The Jump (J) command moves Dot over a specified number of character locations. If no argument is given, the J command moves Dot one character position forward. For example, assume Dot is between characters A and B in the following:

```
A BCDEF
 ↑
 Dot
```

The command 3J moves Dot three characters forward to follow the character, D.

Dot after 3J command

The command -2J moves Dot two characters backward to precede the character, C.

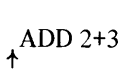


Dot after -2J command

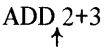
Arguments and their effect upon the Jump command are described as follows:

- nJ Moves Dot forward n characters.
- -nJ Moves Dot backward n characters.
- 0J Moves Dot to the beginning of the current line.
- @J Moves Dot to the location of Mark.
- /J Moves Dot to the end of the buffer.

Carriage return, line feed, form feed, space, and tab characters are counted when executing a J command. For example, assume Dot is at the beginning of the following buffer.



The command 4J moves Dot four characters forward to precede the fifth character, 2. The space between the D and the 2 is counted as one character.



Dot after 4J command.

To move Dot from the beginning of one line to the end of the preceding line, the -2J command can be used to jump backward over the end-of-line characters (CR and LF).

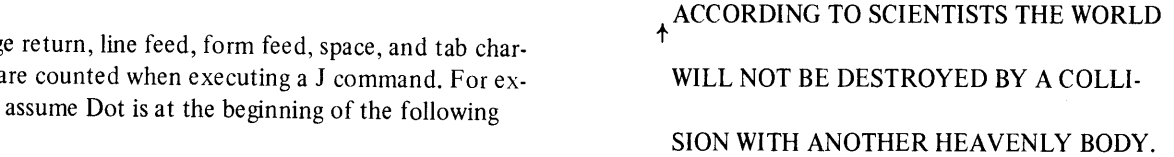
2.3.3 Moving Dot by Line

The Advance (A) command is a line-oriented command that moves Dot to precede the first character of the line determined by the modifier. If no argument is given, EDIT uses one line forward as the argument. The command A moves Dot forward past one end-of-line to precede the first character of the next line. The command 3A moves Dot forward past three ends-of-line to precede the first character of the third line from the current line. An argument preceded by a minus (-) character produces a different result. For example, the command -3A moves Dot backward past three ends-of-line to precede the first character of what is effectively the fourth previous line.

Arguments and their effect upon the Advance command are listed as follows:

- nA Advances Dot n ends of line. Dot is positioned preceding the succeeding line.
- -nA Moves Dot backward to precede n+1 ends of line.
- 0A Moves Dot to the beginning of the current line.
- @A Moves Dot to the location of Mark.
- /A Moves Dot to the beginning of the line following the last line in the buffer.

Assume Dot is at the beginning of the following buffer.



The command 2A moves Dot two ends-of-line to precede the third line.

Dot after 2A command. A -2A command at this point would move Dot to its original location (the beginning of the buffer).

2.3.4 Temporary Reference Point

The Mark (M) command sets the current location of Dot as the temporary reference point Mark. The user can subsequently move the location of Dot and refer to the intervening text with the commercial at (@) character as the modifier. Only one location at a time can be saved as Mark. The format of the Mark command is:

M

with no arguments. For example, assume Dot is at the beginning of the following buffer:

ACCORDING TO LEADING SCIENTISTS,

THE WORLD IS ROUND.

① *M
② *G/,/
③ *@D

① Sets Mark at the current position of Dot.

② Moves Dot to follow the comma (,) character but Mark is left at the beginning of the buffer.

③ Deletes the character string between Dot and Mark. The buffer now reads:

THE WORLD IS ROUND.

Table 2-1 summarizes the commands which can use the Mark @ argument.

Table 2-1
Commands Used With Mark Argument

Command	Result
@A	Moves Dot to the location of Mark.
@C/xxxx/	Changes the characters between Dot and Mark to the specified text.
@D	Deletes the characters between Dot and Mark.
@J	Moves Dot to Mark.
@K	Kills the lines between Dot and Mark.
@L	Lists the lines between Dot and Mark.
@W	Writes the lines between Dot and Mark to the primary output file.
@X/xxxx/	Exchanges the lines between Dot and Mark to the specified text.

CHAPTER 3

CHANGING TEXT STORED IN BUFFER

Several commands can be used to delete and insert characters and lines of text in the buffer. This chapter describes each command and gives an example of each.

3.1 DELETING CHARACTERS

The Delete (nD) command is a character-oriented command that deletes n characters in the page buffer beginning at Dot. If n is not specified, EDIT deletes the character immediately following Dot. Upon completion of a D command, EDIT positions Dot at the first character following the deleted text.

The following list describes each argument and its effect on the Delete command.

- nD — Deletes n characters following Dot. Dot is placed at the first character following the deleted text.
- -nD — Deletes n characters preceding Dot. Dot is positioned at the first character following the deleted text.
- 0D — Deletes the current line up to Dot. Dot is positioned at the first character following the deleted text.
- @D — Deletes the text between Dot and Mark. Dot is positioned at the first character following the deleted text.
- /D — Deletes the text between Dot and the end of the buffer. Dot is positioned at the end of the buffer.

For example, assume Dot is at the beginning of the following buffer:

FOUR SCORE AND SEVEBN YEARS AGO
↑

①*G/VE/ ② *D

① Positions Dot after the VE in the word, SEVEBN.

② Deletes the B after Dot. The line then reads:
FOUR SCORE AND SEVEN YEARS AGO.
↑

3.2 DELETING LINES OF TEXT

The Kill (nK) command removes n lines of text (including the carriage return and line feed characters), from the page buffer beginning at Dot and ending with the nth end-of-line. Dot is placed at the beginning of the line following the deleted text. The following list describes each argument and its effect upon the Kill command.

- nK — Removes the character string (including the CR and LF sequence) beginning at Dot and ending at the nth end-of-line.
- -nK — Removes the character string beginning at the nth end-of-line preceding Dot and ending at Dot. Thus, if Dot is at the center of a line, the modifier -1 deletes one and one-half lines preceding Dot.
- 0K — Removes the current line up to Dot.
- @K — Removes the characters bounded by Dot and Mark.
- /K — Removes the characters beginning at Dot and ending with the last line in the page buffer.

For example, assume DOT is at the beginning of the following buffer:

```
10  READ A,B,C
20  PRINT A;B;C
30  PRINT A,B,C
```

① *2A
② *K

① Advances Dot past two ends-of-line to precede the third line.

② Deletes the third line. The buffer now reads:

```
10  READ A,B,C
20  PRINT A;B;C
```

3.3 INSERTING TEXT

The Insert (I) command is the basic command for inserting text. The specified text is inserted at Dot and Dot is placed after the last character of the inserted text.

If Mark was located following the text to be inserted, Dot becomes the new Marked location.

No arguments can be specified with the Insert command. When EDIT is in Text mode, up to 80 characters per line can be specified by typing the mnemonic I on one line followed by the CR key and the text to be inserted on the following line(s). Execution of the command occurs when the LINE FEED key is pressed. If the text does not contain carriage return or line feed characters, it can be typed on the same line as the I command but must be set off by delimiters. The following are examples of using the I command in Command mode and in Text mode.

```
INPUT B,C, D
      ↑
```

```
*I/A,/ < CR >
*
```

Inserts the characters (A,) at the location of Dot. The buffer then reads:

or

```
*I < CR >
A, < LF >
*
```

```
INPUT B,A, C,D
           ↑
```

A line feed character is inserted in the text only if it is typed as the first character following the I command in text mode. The following example shows how to insert a line feed in the text. (The example assumes EDIT is conditioned to treat a line feed as a BASIC-PLUS continuation line.)

```
10 A,B,C = 10 :PRINT A;B;C
           ↑
```

```
① *I < CR >
    < LF >< tab >
    < LF >
② *–LL
```

① The I command followed by the CR places EDIT in Text mode. A LF character is typed. EDIT enters it in the text. The second LF typed terminates the Insert. (The Tab character improves readability.)

② The two L commands print the result as shown:

```
10 A,B,C = 10
    :PRINT A;B;C
```

EDIT notifies the user of depleted buffer space by reprinting the I command followed by a ? character. For example:

```
*I < CR >
(text to be inserted)

I?
*
```

All text up to but not including the last typed line is in the buffer, but the buffer is almost full. A solution is to create smaller pages by using the F command, ending the editing session, and restarting it for further editing.

3.4 CHANGING CHARACTERS

The Change (nC) command changes a specified number of characters following Dot. A C command is equivalent to an Insert command followed by a Delete command. EDIT requires a text object to be inserted following the nC command. Arguments and their effect upon the C command are described as follows:

- **nC** Replace n characters following Dot with the specified text. Dot is placed after the inserted text.
- **-nC** Replaces n characters preceding Dot with the specified text. Dot is placed after the inserted text.
- **0C** Replaces the current line up to Dot with the specified text. Dot is placed after the inserted text.
- **@C** Replaces the text between Dot and Mark with the specified text. Dot is placed after the inserted text.
- **/C** Replaces the text beginning at Dot and ending with the last character in the buffer. Dot is placed after the inserted text.

For example, assume Dot is at the beginning of the following buffer:

```
↑MEN ARE ALL CREATED
 EQUAL
```

```
*11C/ALL MEN ARE/
```

Changes the first 11 characters to ALL MEN ARE. The buffer now reads:

```
ALL MEN ARE CREATED
EQUAL       ↑
```

Dot after the 11C command.

The C command can be used in Text mode as shown in the following example:

```
ALL MEN ARE CREATED
EQULA
     ↑
```

```
*4C<CR>
AL AND ENDOWED BY <CR>
THEIR CREATOR WITH <CR>
CERTAIN INALIENABLE RIGHTS. <CR >
<LF>
*
```

Changes the characters LA and the CR and LF sequence to the text shown.

3.5 EXCHANGING LINES OF TEXT

The Exchange (nX) command is similar to the Change command except that lines of text, instead of a specified number of characters, are changed. The nX command is identical to an Insert command followed by an nK command. Arguments and their effect upon the Exchange command are listed as follows:

- nX Replaces n lines including the carriage return and line feed characters following Dot. Dot is positioned after the inserted text.
- -nX Replaces n lines including the carriage return and line feed characters preceding Dot. Dot is positioned after the inserted text.
- 0X Replaces the current line up to Dot with the specified text. Dot is positioned after the specified text.
- @X Replaces the lines including the carriage return and line feed characters with the specified text. Dot is positioned after the inserted text.
- /X Replaces the text beginning at Dot and ending with the last character in the buffer with the specified text. Dot is positioned after the inserted text.

The Exchange command can be used in Command mode or Text mode, depending on whether the end-of-line sequence is to be removed or preserved. For example, to remove the end-of-line sequence, use Command mode as shown in the following example:

```
NOW    EW
      ↑
ARE ENGAGDE
```

```
① *X/WE   /
  *V
② NOW WE ARE ENGAGDE
         ↑
```

① Exchanges the text EW and the CR and LF characters with the specified text. This has the effect of joining the two lines.

② Verifies the new line. Dot follows the new text.

To maintain the end-of-line sequence, using Text mode, replace the CR and LF characters, which are removed by the X command as shown in the following example:

```
                NOW WE ARE ENGAGDE
                          ↑
① *X <CR>
  ED<CR>
  IN A GREAT CIVIL WAR<CR>
  <LF>
② *B/L
  NOW WE ARE ENGAGED
  IN A GREAT CIVIL WAR
  *
```

① Exchanges the text DE <CR> with the text ED<CR>IN A GREAT CIVIL WAR<CR>, and maintains the current line sequence.

② Dot is positioned at the beginning of the buffer and the buffer is listed on the user's terminal.

If the user attempts to insert more characters than the page buffer can hold, data from the last line may be lost and text removal does not occur. To avoid this situation separately execute a Kill command followed by an Insert command.

3.6 OPENING SECONDARY INPUT FILE

The Edit Open (EO) command closes the secondary input file and reopens it at the beginning. Modifiers may not be specified with an EO command. An EO command allows the user to make many passes through the secondary input file; however, EDIT allows only one pass per job through the primary output file. An EO command has no effect on the text.

The EO command is useful following an EP or EH command. The following is an example of performing a search through the secondary input file for the first occurrence of the specified text object, killing the contents of the buffer, and reopening the secondary input file at the beginning. Each comment refers to the appropriately marked lines in the example.

```
#ANG<PTU,ZMU

①*EP/THREE/
②*B/L
  SECONDARY INPUT FILE - PAGE THREE
③*/K
④*EO
⑤*ER/L
  SECONDARY INPUT FILE - PAGE ONE



*
```

① A search through the secondary input file is made for the first occurrence of the specified text object. EDIT positions the page containing the text object in the buffer and places Dot after the searched text object. All previously searched pages are deleted and the unsearched pages of the secondary input file are not transferred to the buffer.

② Dot is positioned at the beginning of the buffer and the buffer is listed on the terminal.

③ The contents of the buffer are deleted.

④ The secondary input file is reopened at the beginning.

⑤ The first page of the secondary input file is read into the buffer and the contents of the buffer are listed on the terminal.

3.7 SAVE BUFFER

Text can be stored in an external buffer, called a save buffer, and can subsequently be inserted in several places in the text. The Save (nS) command copies n lines beginning at Dot into the save buffer. The S command operates only in the forward direction; therefore, negative integers cannot be used. Any previous contents of the save buffer are destroyed; however, EDIT does not change the location of Dot nor does it change the current saved data.

If the user specifies more characters than the save buffer can hold, EDIT reprints the related S command followed by a question mark (?) character; none of the specified text is saved.

The Save command is useful to move blocks of text or to insert blocks of text in several places.

The Unsave (U) command inserts the contents of the save buffer at the location of Dot in the main buffer. EDIT places Dot after the last character of the unsaved text. Arguments are not accepted with the U command.

EDIT does not destroy the contents of the save buffer following a U command. Text in the save buffer can be unsaved as many times as desired.

If the data being unsaved contains more characters than the page buffer will hold, EDIT notifies the user by reprinting the related U command followed by a question mark (?) character. EDIT does not execute the U command in this instance.

The U command can be used to move blocks of text or to insert the same block of text in several places.

The following is an example of using the S and U commands. Each comment refers to the appropriately marked lines in the example.

```
#FILA<FILA

①*R/L
 THIS IS AN EXAMPLE OF THE
 SAVE AND UNSAVE COMMANDS.
 PLEASE READ THE FOLLOWING.
②*2S
③*2K
④*>A
⑤*U
⑥*B/L
 PLEASE READ THE FOLLOWING.
 THIS IS AN EXAMPLE OF THE
 SAVE AND UNSAVE COMMANDS.
⑦*EX

#
```

① The primary input file, FILA.BAS, is read into the buffer and the contents of the buffer are listed on the terminal.

② The first two lines of the buffer are transferred to the save buffer.

③ The first two lines of the buffer are then deleted from the buffer. Deleting the lines prevents EDIT from transferring them twice to the output file.

④ Dot is advanced to the end of the buffer.

⑤ The contents of the save buffer are inserted at Dot.

⑥ Dot is returned to the beginning of the buffer and the entire buffer is listed on the terminal.

⑦ The buffer is transferred to the output file, FILA.BAS, and the editing session is ended.

3.8 EXECUTE MACRO COMMAND

Upon receiving an Execute Macro (nEM) command, EDIT executes n times the contents of the save buffer, prior to a carriage return character. To execute a macro (a specified sequence of EDIT commands), the user must insert it in the page buffer, save it, and then execute it. For example, the following commands change every reference to the word MODE to MADE. Each comment refers to the appropriately marked lines in the example.

```
#DEREK<DEREK

①*R/L
 CHARLIE IS A SELF-MODE
 MAN.  IN HIS LIFETIME
 HE HAS MODE OVER $1M.
 ACCORDING TO A FRIEND,
 HE MODE A KILLING IN
 THE MARKET.
②*I
 G/MODE/-3JC/A/

③*BSK3EM
④*B/L
 CHARLIE IS A SELF-MADE
 MAN.  IN HIS LIFETIME
 HE HAS MADE OVER $1M.
 ACCORDING TO A FRIEND,
 HE MADE A KILLING IN
 THE MARKET.
⑤*EX

#
```

① Reads the first page of the file, DEREK.BAS, into the buffer and lists the contents of the buffer on the terminal.

② Inserts the macro in the buffer.

③ Places Dot at the beginning of the buffer. Stores the macro in the save buffer and deletes it from the buffer. Executes the macro three times.

④ Places Dot at the beginning of the buffer and lists the contents of the buffer on the terminal.

⑤ Transfers the contents of the buffer to the output file and ends the editing session.

CHAPTER 4

OUTPUTTING EDITED TEXT

4.1 TERMINATING EDIT

To terminate an editing session, type the EX command, which writes the buffer to the primary output file, transfers the remainder of the primary input file to the output file, closes all open files, and renames the temporary file as the edited, primary output file. The following dialogue shows the procedure:

```
*EX
#^Z
```

EDIT prints the # character (in response to which further input and output specifications can be typed.) The CTRL/Z combination terminates the EDIT program and returns control to the system monitor.

The user should periodically terminate the editing session as described above. EDIT operates in such a manner that the primary input file remains intact during the editing session. The temporary file EDITnn.TMP retains the revised text as edits are made. If the system crashes during the editing session, the primary input file is not disturbed, but the edits being made are lost. Therefore, it is recommended that editing sessions be terminated frequently to update the primary output file. In the event that a system crash occurs, the amount of editing lost is limited to those edits made since the beginning of the latest session.

4.2 WRITING LINES OF TEXT TO OUTPUT FILES

The Write (W) and Edit Write (EW) commands copy lines of text from the buffer to the output files. The Write (W) command copies lines of text from the buffer and appends them to the primary output file. An Edit Write (EW) command is the same as a W command except that EDIT writes to the secondary output file rather than the primary output file. The contents of the buffer are not altered and Dot and Mark are left unchanged.

The argument given with the W and EW commands determines the lines of text to copy. Arguments and their effect upon the Write and Edit Write commands are listed as follows:

nW	Writes n lines of text beginning at Dot and ending with n end-of-line characters to the primary output file.
nEW	Writes n lines of text to the secondary output file beginning at Dot and ending with n end-of-line characters.
-nW	Writes n lines of text to the primary output file beginning at Dot and ending with n+1 p previous end-of-line characters.
-nEW	Writes n lines of text to the secondary output file beginning at Dot and ending with n+1 previous end-of-line characters.
0W	Writes to the primary output file the current line up to Dot.
0EW	Writes to the secondary output file the current line up to Dot.
@W	Writes to the primary output file the text between Mark and Dot.
@EW	Writes to the secondary output file the text between Mark and Dot.
/W	Writes to the primary output file the text between Dot and the end of the buffer.
/EW	Writes to the secondary output file the text between Dot and the end of the buffer.

Write and Edit Write commands can also be used to write portions of text in the buffer to different files. For example, the following set of commands transfers portions of the primary input file to the primary and secondary output files. Each comment refers to the appropriately marked lines in the example.

```
#FIL1,FIL2<WALTER

① *R/L
  10FOR I=0 TO 5
  15READ A(I)
  20NEXT I
  25DATA 5,65,66,67,68
② *2A
③ *-2W
④ *2EW
⑤ *B/K
⑥ *EX

#
```

① Reads the primary input file, WALTER.BAS, into the buffer and lists the contents of the buffer on the terminal.

② Advances Dot two end-of-line characters to precede the third line.

③ Writes the two lines preceding Dot into the primary output file, FIL1.BAS.

④ Writes the two lines following Dot into the secondary output file, FIL2.BAS.

⑤ The B command moves Dot to precede the first character in the buffer. The /K command removes the characters beginning at Dot and ending with the last character in the buffer. This action prevents unwanted text from being transferred to the primary output file. Text in the primary input file is not destroyed.

⑥ Closes all open files and terminates the editing session. EDIT is ready to accept another input/output specification.

4.3 CLOSING PRIMARY FILES ONLY

The End File (EF) command closes the primary input and output files. Further input and output using the primary files is not allowed by EDIT; however, secondary output and input files remain open for further editing.

The EF command can be used to create an output file from a section of a large input file. Modifiers cannot be specified with an EF command.

The following is an example of the EF command. Each comment refers to the appropriately marked lines in the example.

```
#ELLEN,TERRY<MARY,BRENDA

① *R/L
  NOW IS THE TIME
  FOR ALL GOOD MEN
  TO COME TO THE AID
  OF THEIR COUNTRY.
② *ER/L
  NOW IS THE TIME
  FOR ALL GOOD MEN
  TO COME TO THE AID
  OF THEIR COUNTRY.
  FOUR SCORE AND SEVEN
  YEARS AGO,
③ *#W
④ *EF
⑤ *R/L
```

① The first page of the primary input file, MARY.BAS, is read into the buffer and the contents of the buffer are printed on the terminal.

② The first page of the secondary input file, BRENDA.BAS, is read into the buffer and is appended to the current contents. The buffer is listed on the terminal.

③ The contents of the buffer are written to the primary output file, ELLEN.BAS.

④ The primary input file and the primary output file are closed.

⑤ An attempt is made to read the next page of the primary input file after specifying the EF command. EDIT responds with the related R command followed by a question mark (?).

(continued on next page)

(continued from previous page)

```
   R?
⑥ *4EW
⑦ *EX

   #
```

⑥ The first four lines of the buffer are written to the secondary output file, TERRY.BAS.

⑦ The editing session is ended.

APPENDIX A

EDIT EXAMPLES

This appendix supplies three examples of using EDIT. Each example is comprised of a listing and some explanatory comments relative to specific lines of printout. Example 1 illustrates creating a file, terminating EDIT, and reopening the file for further editing. Example 2 shows how to edit an existing file without having to specifically enter it into the buffer. Example 3 illustrates the merging of two input files into one output file.

Example 1:

```
① RUN $EDIT
  EDIT V05B-11

  #MOSQ

② *I
  THE MOSQUETO IS ONE OF MANS MOST ANCIENT AND DEADLY
  ENEMIES MORE THAN 225 VIRUSES, 13 OF THEM POTENTAILLY
  FATAL TO HUMAN BEINGS ARE KNOWN TO BE CARRIED BY
  THIS INCREDIBLY ADAPABLE PEST, WHOSE 2000 SPECIESS
  BREED BY THE BILLIONS ALMOST IN EVERY PART OF THE WORLD.

③ *EX

  #^Z

  READY

④ RUN $EDIT
  EDIT V05B-11

⑤ #MOSQ<MOSQ

⑥ *R
⑦ *L
  THE MOSQUETO IS ONE OF MANS MOST ANCIENT AND DEADLY
⑧ *9JC/I/
⑨ *2G/N/I//
⑩ *V
  THE MOSQUITO IS ONE OF MAN'S MOST ANCIENT AND DEADLY
⑪ *AL
  ENEMIES MORE THAN 225 VIRUSES, 13 OF THEM POTENTAILLY
⑫ *G/S/I/./
⑬ *4G/T/2C/IA/V
  ENEMIES. MORE THAN 225 VIRUSES, 13 OF THEM POTENTIALLY
⑭ *AL
  FATAL TO HUMAN BEINGS ARE KNOWN TO BE CARRIED BY
```

```
⑮ *G/S/I/,/
⑯ *V
  FATAL TO HUMAN BEINGS, ARE KNOWN TO BE CARRIED BY
⑰ *AL
  THIS INCREDIBLY ADAPABLE PEST, WHOSE 2000 SPECIESS
⑱ *G/P/I/T/
⑲ *4G/S/
⑳ *D
㉑ *V
  THIS INCREDIBLY ADAPTABLE PEST, WHOSE 2000 SPECIES
㉒ *AL
  BREED BY THE BILLIONS ALMOST IN EVERY PART OF THE WORLD.
㉓ *4G/ /9C/IN ALMOST/
㉔ *V
  BREED BY THE BILLIONS IN ALMOST EVERY PART OF THE WORLD.
㉕ *EX

㉖ #^Z

  READY
```

The following description explains how in Example 1, a file is created, EDIT is terminated, and the file is reopened for further editing. Each comment refers to the appropriately marked lines in the example.

1. The user loads EDIT and creates the output file, MOSQ.BAS. The keyboard is the default input device.
2. The Insert command followed by a carriage return (CR) character instructs EDIT to enter Text mode. Lines of text are typed and the CR key is pressed following each line. The line feed (LF) character following the text to be inserted instructs EDIT to insert the text into the buffer.
3. EDIT returns to Command mode as indicated by the asterisk (*) character printed on the terminal. The user can then edit the file by typing appropriate commands. If no further editing is desired, the file is closed by typing the Exit command. The CTRL/Z combination terminates the editing session and returns control to the system monitor.
4. To rerun EDIT, type RUN $EDIT. The version number of EDIT is then printed on the user's terminal.
5. EDIT prints a number sign (#) character and waits for the user to specify an input and an output file. EDIT searches for the file MOSQ.BAS in the user's current account and opens it for input. Because the filenames of the primary input and primary output files are the same, EDIT creates the backup file with a .BAK extension.
6. When the file has been found, EDIT responds with an asterisk (*), indicating Command mode. The R command reads the first page of the input file into the buffer. EDIT positions Dot at the beginning of the buffer.
7. The L command lists the first line of the buffer on the terminal.
8. The 9J command moves Dot over nine characters to precede the E character in MOSQUETO. The C command changes the following character to I. Dot resides after the I character. The slash (/) is used as a delimiter because it does not appear in the text object.
9. The 2G command locates the second occurrence of N and the I command inserts an apostrophe (') after it. Dot is placed after the apostrophe character.
10. The V command verifies the edited line.
11. The AL combination advances Dot to the beginning of the next line and lists the line on the terminal.
12. The G command finds the first occurrence of S and EDIT positions Dot immediately after it. The I command inserts a period (.) at Dot. Dot is placed following the period (.) character.
13. The 4G command moves Dot to follow the fourth occurrence of the character T. The 2C command changes the next two characters, AI, in POTENTAILLY, to IA for POTENTIALLY. Dot is placed after the inserted text. The line is then verified.

14. The AL combination advances Dot one line and lists the line on the terminal.
15. The user inserts a comma (,) after BEINGS by using the G command to search for the first occurrence of S, and the I command to insert a comma (,) character following S. Dot is positioned after the inserted text.
16. The V command verifies the edited line.
17. Dot is advanced one line and the line is listed on the terminal.
18. The user inserts a T between the characters P and A in ADAPABLE by using the Get and Insert commands. The G command positions Dot to follow the P. The I command inserts a T at Dot. EDIT positions Dot after the inserted T.
19. The Get command searches for the fourth occurrence of S following Dot and positions Dot after it.
20. The Delete command deletes the character following Dot.
21. The V command prints the edited line on the terminal.
22. The AL combination advances Dot to precede the next line and lists the line on the terminal.
23. The character string ALMOST IN is changed to IN ALMOST by a Get and a Change command. The fourth space character is found and Dot is placed following it. The 9C command changes the following nine characters to IN ALMOST. EDIT positions Dot after the T in ALMOST.
24. The V command prints the edited line on the terminal.
25. The EX command ends the editing session.
26. The CTRL/Z combination terminates EDIT and returns control to the system monitor. RSTS responds with a READY message.

Example 2:

```
① EDIT DRUG


② *L
   UNLESS MOSQUITOES ARE CONTROLLED, DEADLY EPIDEMICS CAN RESULT.
③ *EX

   READY
```

The following description explains how in Example 2, the user can load the EDIT program and read the first page of the primary input file by specifying only one command. This feature is optional and is not available in all RSTS/E systems. Each comment refers to the appropriately marked lines in the example.

1. Typing the CCL command, EDIT, and the filename on the same line loads the EDIT program, opens the file DRUG.BAS for input, and reads the first page into the buffer.
   The R command is not needed.
2. When the first page has been read into the buffer, EDIT responds with an asterisk (*) character indicating Command mode. Dot resides at the beginning of the buffer. The L command lists the first line of the buffer on the terminal.
3. EDIT returns to Command mode as indicated by the * character. The EX command terminates the editing session and returns control to the system monitor.

NOTE

This method of running EDIT is more efficient than Example 1 for two reasons:

1. The input file is automatically read into the buffer; therefore, the R command is not necessary.
2. The EX command automatically restores the system monitor after closing the file. The CTRL/Z combination is not needed.

Example 3:

```
   RUN $EDIT
   EDIT V05B-11

   #DRUG2<MOSQ,DRUG

①  *R
②  *^/L
   THE MOSQUITO IS ONE OF MAN'S MOST ANCIENT AND DEADLY
   ENEMIES.  MORE THAN 225 VIRUSES, 13 OF THEM POTENTIALLY
   FATAL TO HUMAN BEINGS, ARE KNOWN TO BE CARRIED BY
   THIS INCREDIBLY ADAPTABLE PEST, WHOSE 2000 SPECIES
   BREED BY THE BILLIONS IN ALMOST EVERY PART OF THE WORLD.
③  *5AV
④  *ER
⑤  *^/L
   THE MOSQUITO IS ONE OF MAN'S MOST ANCIENT AND DEADLY
   ENEMIES.  MORE THAN 225 VIRUSES, 13 OF THEM POTENTIALLY
   FATAL TO HUMAN BEINGS, ARE KNOWN TO BE CARRIED BY
   THIS INCREDIBLY ADAPTABLE PEST, WHOSE 2000 SPECIES
   BREED BY THE BILLIONS IN ALMOST EVERY PART OF THE WORLD.
   UNLESS MOSQUITOES ARE CONTROLLED, DEADLY EPIDEMICS CAN RESULT.
⑥  *EX

   #
```

The following description explains how in Example 3 two input files are merged into one output file. Each comment refers to the appropriately marked lines in the example.

1. The first page of the primary input file, MOSQ.BAS, is read into the input buffer.
2. The /L command lists the contents of the buffer on the terminal.
3. The 5AV combination advances Dot five lines and verifies the next line. Since there is no next line, nothing is printed.
4. The ER command reads the first page of the secondary input file, DRUG.BAS, and appends it to the buffer. EDIT positions Dot at the beginning of the buffer. This has the effect of merging the two pages in the buffer.
5. The /L command lists the entire buffer on the terminal.
6. The EX command writes the contents of the buffer to the primary output file, DRUG2.BAS, transfers any remaining pages of MOSQ.BAS to DRUG2.BAS, closes all files, and terminates the editing session.

APPENDIX B

SUMMARY OF COMMANDS

This appendix is an alphabetic summary of the commands used in the EDIT program. A / represents any legal text delimiter, a <CR> represents a return character, a <LF> represents a line feed character, and an n represents the number of characters or lines to be edited.

COMMAND	FORMAT	RESULTS
Advance	nA	Moves Dot forward past n end-of-line characters to precede the first character of the succeeding line.
	-nA	Moves Dot backward past n end-of-line characters to precede n+1 lines.
	0A	Moves Dot to precede the first character of the current line.
	@A	Moves Dot to the location of Mark.
	/A	Moves Dot to follow the last character in the buffer.
Beginning	B	Moves Dot to precede the first character in the buffer.
Change	nC/xxxx/	Changes n characters following Dot to the text, xxxx. The text object must be set off by delimiters. Moves Dot to follow the last character of xxxx. Equivalent to a Delete command followed by an Insert command.
	nC<CR> xxxx<CR> <LF>	Same as the respective nC command above except multiple lines can be used to replace n characters. In this format, delimiters are not needed before and after the text object. The LF key terminates the insert.
	-nC/xxxx/	Same as respective nC command above except n characters preceding Dot are changed.
Delete	nD	Deletes n characters following Dot. The modifiers 0, @, and / can be used.
Edit Open	EO	Closes the secondary input file and opens it again for input.
Edit Position	nEP/xxxx/ or nEP<CR> xxxx<CR> <LF>	Performs a search through the secondary input file for the nth occurrence of the text object. Dot is placed after the text object. The buffer is deleted upon an unsuccessful search.
Edit Read	ER	Reads the next page of the secondary input file into the buffer. EDIT positions Dot at the beginning of the buffer.

COMMAND	FORMAT	RESULTS
Edit Whole	nEH/xxxx/ or nEH<CR> xxxx<CR> <LF>	Performs a search through the secondary input file for the nth occurrence of the specified text object. Dot is placed after the text object. The contents of the buffer are transferred to the primary output file.
Edit Write	nEW	Writes n lines into the secondary output file. The modifiers -n, 0, @, and / can be used.
End File	EF	Closes the primary output and primary input files to any further input or output.
Exchange	nX/xxxx/ or nX <CR> xxxx <CR> <LF>	Exchanges n lines for xxxx. Equivalent to an Insert command followed by a Kill command. The modifiers -n, 0, @, and / can be used.
Execute Macro	nEM	Executes the first line of the save buffer as a command string n times.
Exit	EX	Writes the text in the buffer to the primary output file, transfers the remainder of the primary input file to the primary output file, closes all files, and returns to Command mode.
Form Feed	F	Writes a form feed after Dot in the buffer.

COMMAND	FORMAT	RESULTS
Get	nG/xxxx/ or nG <CR> xxxx <CR> <LF>	Searches for the nth occurrence of xxxx and positions Dot after it.
Insert	I/xxxx/ or I <CR> xxxx <CR> <LF>	Inserts the text object at Dot. Moves Dot to follow the text object.
Jump	nJ	Moves Dot ahead n characters. The modifiers -n, 0, @, and / can be used.
Kill	nK	Removes n lines of text. The modifiers -n, 0, @, and / can be used.
List	nL	Prints n lines in the buffer on the user's terminal. The modifiers -n, 0, @, and / can be used.
Mark	M	Sets the current location of Dot to the Marked location, which subsequently can be referenced by the @ modifier.
Next	nN	Writes the contents of the buffer to the primary output file, deletes the contents of the buffer, and reads the next page of the primary input file into the buffer. The sequence is repeated until n pages have been read or until the end-of-file has been reached. Dot is positioned at the beginning of each page read into the buffer.

COMMAND	FORMAT	RESULTS
Position	nP/xxxx/ or nP<CR> xxxx<CR> <LF>	Performs a search starting at Dot in the primary input file for the nth occurrence of xxxx. The search is continued until the text object is found or the end of the file is reached. If the search is successful, Dot is positioned after the text object.
Read	R	Reads a page of the primary input file into the buffer. Dot is positioned at the beginning of the buffer.
Save	nS	Inserts n lines into the save buffer.
Unsave	U	Copies the contents of the save buffer into the buffer at Dot. Dot is positioned after the inserted text.
Verify	V	Prints the line on which Dot is located. The location of Dot is not changed.
Whole	nH/xxxx/ or nH <CR> xxxx <CR> <LF>	Performs a search through the primary input file for the nth occurrence of xxxx. Each page of the primary output is searched until the text object is found or the end of the file is reached. If the search is successful, Dot is positioned after the text object.
Write	nW	Writes n lines beginning at Dot in the buffer to the primary output file. The modifiers -n, 0, @, and / can be used.

APPENDIX C
ERROR MESSAGES

Prior to executing any commands, EDIT scans the entire command string for errors in command format (illegal arguments, illegal combinations of commands, etc.). If the error occurs after the first command in a command string, EDIT reprints the command string with a question mark (?) following the character where EDIT found the error. Those commands preceding the command questioned are executed, while those commands following the ? are not executed. For an explanation of a specific error condition detected by EDIT refer to the section in this manual that describes the command questioned.

The following is an example of an error message printed by EDIT:

```
B2G/$/DN?
*
```

EDIT moves Dot to the beginning of the buffer, searches for the second occurrence of the $ character, deletes the character searched, and tries to read the next page into the buffer. EDIT encounters the end of the file and is therefore unable to execute the N command. EDIT returns to Command mode as indicated by the printed *.

Other error messages which could appear while in EDIT are listed in Table C-1.

Table C-1
EDIT Command Error Messages

Message	Meaning
LINE LENGTH OF X IS TOO LONG LINE BELOW WILL BE LOST... text	EDIT reads input and finds line greater than 240 characters and prints this warning message. Reformat the line indicated so that it is less than 240 characters long.
HOW MANY SECONDS TO WAIT FOR 'EW' DEVICE?	An assignable device is specified as a secondary output but is not available for execution of an EW command. If a nonzero number is typed, EDIT sleeps and tries the EW command again. If 0 is typed, EDIT ignores the EW command and prints the * again.
*****IGNORING A TOO LONG LINE*****	EDIT reads input and encounters line too long (ERR=47) error. EDIT cannot recover lines longer than 255 characters.

INDEX

INDEX (Cont.)

INDEX (Cont.)

RSTS/E Text Editor Manual
DEC-11-UTEMA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do require a written reply, please check here. ☐

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.