Price: $3.50

TYMSHARE REFERENCE SERIES

EDITOR

REFERENCE MANUAL

TYMSHARE, INC.
CUPERTINO, CALIFORNIA

TYMSHARE®

FEBRUARY 1978
VERSION 1

CONTENTS

Page

TYMSHARE MANUALS
SYMBOL CONVENTIONS

The symbols used in this manual to indicate Carriage Return, Line Feed, and ALT MODE/ESCAPE are as follows:

Carriage Return: ↵

Line Feed: ↴

ALT MODE/ESCAPE: ⊕ *NOTE: This symbol will be printed as many times as it is required to hit this key.*

Control Characters

Control characters, typed by pressing and holding the CTRL key in conjunction with an alphabetic character, are denoted by a superscript c. For example, D^c denotes Control D.

Action At The Terminal

To indicate clearly what is typed by the computer and what is typed by the user, the following color code convention is used:

Computer: Black **User: Red**

INTRODUCTION

EDITOR provides the user of the Tymshare system with an unusually flexible set of editing commands. The EDITOR user may

- Edit individual lines of text.
- Edit commands as they are typed.
- Insert a line or range of lines into the text.
- Delete a line or range of lines from the text.
- Move a line or range of lines to another position in the text.
- Copy a line or range of lines in the text.
- Make substitutions throughout the text.
- Use most of the commands repetitively without being required to retype them.
- Write small editing programs to do much of his editing work for him.

ABOUT THIS MANUAL

This manual is intended primarily for reference. However, it is also a readable and effective text for the potential EDITOR user.

For those who are beginners at computer aided text editing, we suggest studying Tymshare's *EDITOR Manual, Instant Series*, before this manual is attempted.

Section 1 of this manual is designed to familiarize the reader with the fundamentals of EDITOR, or to serve as a review for those who have already been introduced to EDITOR.

Sections 2 through 5 constitute the principal reference part of the manual. All EDITOR commands are discussed fully in these sections. However, the student of EDITOR will find this part of the manual, especially the examples, easier to understand if he reads Section 1 first.

Section 6 contains editing examples demonstrating various techniques.

The appendices provide a summary of EDITOR's error messages, a convenient table showing the definition of the current line after each EDITOR command, some information about the terminal, and a summary of EDITOR's commands and control characters.

SECTION 1
INTRODUCTION TO EDITOR: AN EXAMPLE

As an introduction to EDITOR we present a complete example of text editing with EDITOR.

We begin with the first steps in using any Tymshare language, logging in and calling the language:

After the connection with the computer has been properly made, the computer replies with:

PLEASE LOG IN ↵	The user types a Carriage Return.
ACCOUNT: A3 ↵	He types his account number (A3 in this case) followed by a Carriage Return.
PASSWORD: ↵	The user types his password followed by a Carriage Return. The password does not print.
USER NAME: JONES ↵	He types his user name.
PROJ CODE: K-123-K ↵	He types a project code (K-123-K in this case). A project code is optional; if no project code is desired, type a Carriage Return in response to the system's request.
TYMSHARE 12/24 11:20 **– EDITOR** ↵	The user is properly connected to the system. The dash (–) on the left indicates that he is in the EXECUTIVE. EDITOR may now be called by typing EDITOR followed by a Carriage Return. EDITOR replies with an asterisk (*) when it is ready to accept a command.
*** APPEND** ↵ **87100 ALBUQUERQUE, NEW MEX** ↵ **98310 BREMERTON WASHINGTON** ↵ **90241 DOWNYA^{C}←EY CALIF** ↵ **98345 KEYPORT WASHINGTON** ↵ **87544 LOS ALAMOS NEW MEX** ↵ **94025 MENLO CALIF** ↵ **00003 PEORIA ILL** ↵ **95112 SAN JOSE CALIF** ↵ **95112 SEATTLE WASHQ^{C}↑** **98124 SEATTLE WASHINGTON** ↵	The APPEND command allows the user to enter his text. He uses the control characters A^{C} and Q^{C} to make corrections as he types.
D^{C}	Control D terminates the APPEND command.
*** /** **87100 ALBUQUERQUE, NEW MEX** **98310 BREMERTON WASHINGTON** **90241 DOWNEY CALIF** **98345 KEYPORT WASHINGTON** **87544 LOS ALAMOS NEW MEX** **94025 MENLO CALIF** **00003 PEORIA ILL** **95112 SAN JOSE CALIF** **98124 SEATTLE WASHINGTON**	The / command prints his entire text on the terminal.

```
* WRITE↵ ..... He saves his text on a disk file, called /ZIP/.
TO /ZIP/↵
 NEW FILE↵
69 WORDS.
* 1EDIT↵ ..... The user edits line 1. Using control characters, he de-
87100 ALBUQUERQUE, NEW MEX                letes the comma.
Z^CR87100 ALBUQUERQUES^C%D^C NEW MEX
* 1/ ..... He prints the new line 1.
87100 ALBUQUERQUE NEW MEX
* 'MENLO'EDIT↵ ..... He edits the line containing the text MENLO and
94025 MENLO CALIF                         prints it.
Z^CO94025 MENLOE^C< PARKE^C>D^C CALIF
* '94025'/
94025 MENLO PARK CALIF
* '0000'DELETE↵ ..... He deletes the line containing the text 0000.
* SUBSTITUTE↵ ..... He abbreviates the word WASHINGTON throughout
"WASHD^C" FOR "WASHINGTOND^C"             the text.
WAIT? N
3
* / ..... He prints the entire edited text.
87100 ALBUQUERQUE NEW MEX
98310 BREMERTON WASH
90241 DOWNEY CALIF
98345 KEYPORT WASH
87544 LOS ALAMOS NEW MEX
94025 MENLO PARK CALIF
95112 SAN JOSE CALIF
98124 SEATTLE WASH
* WRITE↵ ..... The user saves the edited text on the same file he cre-
TO /ZIP/↵                                 ated earlier, /ZIP/.
 OLD FILE↵
58 WORDS.
* QUIT↵ ..... The user leaves EDITOR.
– LOGOUT↵ ..... To leave the system, he uses the EXECUTIVE com-
TIME USED 0:8:03                          mand LOGOUT. After the system types the amount
PLEASE LOG IN:                            of time he has used, he can hang up or let someone
                                          else log in.
```

Before proceeding with a step-by-step analysis of our example, we note a few general characteristics of EDITOR.

EDITOR is an editing language consisting of various editing commands, such as APPEND and SUBSTITUTE. The "data" upon which the commands operate is simply text.

In our example, the text consists of alphabetic and numeric data; namely, ZIP codes and their corresponding cities and states, which the user might wish to use as input to a program in one of Tymshare's programming languages. However, EDITOR itself is not a programming language, but a general purpose text editor. Any text that can be typed on the Tymshare terminal can be accepted by EDITOR. For example, this manual could be typed into EDITOR and updated from time to time.

Once he has called EDITOR, the user has EDITOR's large text area available to him. Text in the text area

may be edited quickly and easily using EDITOR's commands; the edited text may be saved on a disk file, printed at the terminal, or punched on paper tape. In general, the use of EDITOR will usually involve

- Input of text to the text area.
- Editing of the text.
- Output of text from the text area.

Our example shows some ways of doing all of these. We shall now discuss them in more detail, in the order that they appear in the example.

INPUT OF TEXT: THE APPEND COMMAND

Text may be entered into EDITOR's text area from paper tape, from a disk file, or directly from the terminal. In our example, the user wishes to enter his text from the terminal, so he uses the APPEND command. This command has the following form:

*** APPEND ↵**
the text to be entered ↵
D^{C}
*****

After typing APPEND followed by a Carriage Return, the user types his text. The command is terminated by a Control D (D^{C}), which must immediately follow the last Carriage Return of the text. EDITOR will then type the asterisk, indicating that it is ready to accept another command.

CONTROL CHARACTERS

Control characters have many uses in EDITOR. They are typed by depressing and holding the CTRL key and then typing the desired letter, D in the above case. They do **not** print on the terminal; however, EDITOR will make some response when the user types a control character. In the above case, it typed an asterisk to indicate that the D^{C} did in fact terminate the APPEND.

NOTE: Throughout this manual, control characters will be indicated by the superscript c; for example, D^{C} denotes Control D.

Text entered during APPEND may be edited while it is being typed by using control characters.

CONTROL A: DELETING THE PRECEDING CHARACTER

In our example, the word DOWNEY was misspelled as DOWNY in the third line of text. To correct this error, the user types a Control A to delete the preceding character, Y. EDITOR types a back arrow (←) to indicate that Control A is used, thus:

90241 DOWNYA^{C}←EY CALIF ↵

CONTROL Q: DELETING THE PRECEDING LINE

Similarly, in the last line of text entered during APPEND, the user noticed that he had typed the wrong ZIP code at the very beginning of the line. He types a Control Q to delete the entire line. EDITOR types an up arrow (↑) and returns the carriage for the user, who then types the correct line:

95112·SEATTLE WASHQ^{C}↑
98124 SEATTLE WASHINGTON ↵

OUTPUT OF TEXT

THE / COMMAND: PRINTING TEXT AT THE TERMINAL

Very often it is useful to have the contents of the text area printed at the terminal. The command to do this is the /. Unlike other commands, the / acts immediately after it is typed, so there is no need to follow it with a Carriage Return. EDITOR returns the carriage and prints the contents of the text area.

In the example, the user typed a / after he had terminated the APPEND, thus:

```
* /
87100 ALBUQUERQUE, NEW MEX
98310 BREMERTON WASHINGTON
90241 DOWNEY CALIF
98345 KEYPORT WASHINGTON
87544 LOS ALAMOS NEW MEX
94025 MENLO CALIF
00003 PEORIA ILL
95112 SAN JOSE CALIF
98124 SEATTLE WASHINGTON
*
```

Note that there is no evidence of the control characters used during APPEND. Printing the contents of the text area after APPEND can be especially useful when many control characters have been used.

THE WRITE COMMAND: SAVING TEXT ON A FILE

After printing the contents of the text area, the user decides to save his text on a file, even though he has more editing to do. This practice assures him that his text will always be safe. Even if he accidentally destroys some of it, it can always be read back into the text area from the disk file.

The command to write text on a file is

WRITE ↵

After this is typed, EDITOR responds with

TO

The user then types the name of the file on which he wishes to save the text, followed by a Carriage Return. The file name may consist of any characters beginning and ending with slashes. *NOTE: The name may not contain a slash except as the first and last character.*[1] In our example we have the file name

/ZIP/

If the user does not already have a file with the chosen file name in his directory, EDITOR responds with

NEW FILE

The user may then type a Carriage Return, telling EDITOR to save the text on the file, or he may press the ALT MODE or ESCAPE key to abort the command.

If the user does have a file with the chosen name, EDITOR will respond with

OLD FILE

and again, the command can be confirmed with a Carriage Return or aborted with an ALT MODE or ESCAPE. If the command is confirmed, the text will be written on the file and whatever was previously in the file will be lost.

After a WRITE command has been confirmed, EDITOR prints the number of words written on the file (3 characters = 1 word) and returns control to the user. Thus, in our example we have

```
* WRITE ↵
TO /ZIP/ ↵
 NEW FILE ↵
69 WORDS.
*
```

EDITING TEXT IN THE TEXT AREA

The various editing commands available to the EDITOR user enable him to manipulate text in many ways. In our example the user edits individual lines of text, deletes a whole line of text, and makes substitutions throughout the text.

LINE ADDRESSING

To edit a line of text, or to manipulate whole lines of text, a method of telling EDITOR which line or lines are to be changed is needed. This is accomplished

1 - Other kinds of file names are also allowed. These are discussed under *Rules For Naming Files*, Page 21.

by various kinds of line addressing. In our example, two methods of line addressing are used:

1. Addressing a line by its EDITOR line number.
2. Addressing a line by text within the line.

EDITOR assigns a line number to each line of text. The line numbers are always consecutive integers beginning with 1. Thus, to edit the first line of text, the user types

1EDIT ↵

When he wishes to edit the line containing the word MENLO, the user finds it easier to address the line by text within the line, so he types

'MENLO'EDIT ↵

THE EDIT COMMAND

This command is used to make changes within a line of text. It has the form

*a***EDIT** ↵

where *a* is an address of the line to be edited. After the Carriage Return following the command, EDITOR prints the addressed line, called the old line, and returns the carriage. To change the line, the user may simply retype the new line below the old line and end the edit with a Carriage Return. However, control characters may be used to edit more efficiently. The first such use of control characters in our example is in the edit of line 1:

```
* 1EDIT ↵
87100 ALBUQUERQUE, NEW MEX
Z^CR87100 ALBUQUERQUES^C%D^C NEW MEX
*
```

First, a Control Z is typed, followed by the character R, to copy the entire line up to and including the R. EDITOR copies the specified characters. Then the user types the remaining three characters preceding the comma (Q, U, and E). He types a Control S to delete the corresponding character in the old line, the comma. A Control D copies the rest of the old line to the new line and ends the edit.

After editing the line, the user prints it to see what it now looks like:

```
* 1/
87100 ALBUQUERQUE NEW MEX
*
```

An example of inserting text with control characters is shown in the next use of the EDIT command:

```
* 'MENLO'EDIT ↵
94025 MENLO CALIF
Z^CO94025 MENLOE^C< PARKE^C>D^C CALIF
* '94025'/
94025 MENLO PARK CALIF
*
```

Again, Control Z is used to copy up to and including the character typed after it, an O in this case. Then Control E is used to insert a space and the word PARK, as follows: First, the user types a Control E; EDITOR responds with a left angle bracket (<). Then a space followed by the word PARK is typed. Another Control E terminates the insert; this time EDITOR responds with a right angle bracket (>). Again, Control D copies the rest of the old line and ends the edit.

The command

'94025'/

prints the entire edited line.

THE DELETE COMMAND

To delete a line of text, the command

*a***DELETE** ↵

is used, where *a* is an address of the line to be deleted. Thus, to delete the line

00003 PEORIA ILL

in the example, the user types

'0000'DELETE ↵

THE SUBSTITUTE COMMAND

The last editing done in the example is abbreviating the word WASHINGTON everywhere it occurs in the text. To do this, the SUBSTITUTE command is used:

```
* SUBSTITUTE ↵
"WASHD^C" FOR "WASHINGTOND^C"
WAIT? N
3
*
```

First, the user types SUBSTITUTE followed by a Carriage Return. EDITOR then prints a double quote. The user types the characters to be inserted, WASH, terminated by a Control D. EDITOR prints

" FOR "

to prompt the user. Now he types the characters to be replaced, WASHINGTON, again terminated by a Control D. EDITOR prints another double quote, returns the carriage, and asks the question

WAIT?

In our example, the user types an N (for No, do not wait).[1] There is no need to type a Carriage Return after the N; EDITOR returns the carriage, makes the substitutions, and prints the number of substitutions made, 3.

Having abbreviated the word WASHINGTON, the user again prints his text:

```
* /
87100 ALBUQUERQUE NEW MEX
98310 BREMERTON WASH
90241 DOWNEY CALIF
98345 KEYPORT WASH
87544 LOS ALAMOS NEW MEX
94025 MENLO PARK CALIF
95112 SAN JOSE CALIF
98124 SEATTLE WASH
*
```

He decides he has no more editing to do, so he saves his text on the file /ZIP/. Note that EDITOR responds with OLD FILE this time:

```
* WRITE ↵
TO /ZIP/ ↵
 OLD FILE ↵
58 WORDS.
*
```

QUIT and CONTINUE

To leave EDITOR and return to the EXECUTIVE, the user types

```
* QUIT ↵

-
```

The EXECUTIVE dash indicates that he is no longer in EDITOR. At this point, he may call another language, do some work in the EXECUTIVE, or log out. If he does not call another language or a library program (or log out), he may return to EDITOR by typing the EXECUTIVE command

```
- CONTINUE ↵
```

and his text will still be in the text area. If EDITOR is recalled by typing EDITOR again, the text will no longer be there.

CONVENTIONS FOR TYPING COMMANDS

Any of the EDITOR commands may be shortened. For example, the APPEND command can be typed in any of the following ways:

```
APPEND
APPEN
APPE
APP
AP
A
```

NOTE: TAPE and PUNCH can be abbreviated to no less than the first two letters. All other commands can be abbreviated to one letter.

Spaces are **not** ignored when typing commands. For example,

```
APP END ↵
```

and

```
A ↵
```

result in a question mark. Spaces between the parts of a command form are allowed, however. Thus

```
3 EDIT ↵
```

is legal.

1 - If WAIT? is answered with a Y, EDITOR allows the user to make selective substitutions. See *Repetitive Editing*, Page 33.

SUMMARY OF SECTION 1

COMMAND SUMMARY

The following EDITOR commands were introduced in this section:

Command Type	Command Name	Command Function
Input Commands	APPEND	Enters text into text area from terminal.
Editing Commands	DELETE	Deletes entire lines of text.
	EDIT	Allows edit of lines.
	SUBSTITUTE	Allows substitution of characters throughout text.
Output Commands	/	Prints text at terminal.
	a/	Prints the line addressed by *a*.
	WRITE	Writes text on a file.
Utility Commands	QUIT	Returns to EXECUTIVE.

SUMMARY OF CONTROL CHARACTERS

The following control characters were introduced in this section:

Control Character	Function
A^C	Deletes previous character in line being typed.
Q^C	Deletes last line typed.
D^C	During EDIT, copies rest of old line to new line and ends EDIT. Also used to terminate APPEND.
E^C	Inserts text into old line.
S^C	Deletes next character in old line.
Z^C *and a character*	Copies old line to new line up to and including character typed after it.

SECTION 2

INPUT, OUTPUT, AND ADDRESSING OF TEXT

INPUT OF TEXT TO TEXT AREA

Text may be entered into EDITOR's text area from the terminal, from paper tape, or from a file. The following commands are used for these purposes:

<table>
<tr><th>Type Of Input</th><th colspan="2">General Form Of Command</th><th>Command Function</th></tr>
<tr><td>Terminal Input</td><td colspan="2">APPEND ↵
text to be entered ↵
D^C</td><td>Text typed on the terminal is brought into the text area.</td></tr>
<tr><td>Paper Tape Input</td><td colspan="2">TAPE ↵
The user turns on the paper tape reader. When text is read, he turns the reader off and types
D^C</td><td>Text on the tape is brought into the text area and also printed on the terminal.</td></tr>
<tr><td rowspan="3">Disk File Input</td><td>Long Form</td><td>Short Form</td><td rowspan="2">Text in the file is brought into the text area.</td></tr>
<tr><td>READ ↵
FROM: file name ↵</td><td>READ file name ↵</td></tr>
<tr><td>aREAD ↵[1]
FROM: file name ↵</td><td>aREAD file name ↵</td><td>Text in the file is inserted into the text area before the line addressed by a.</td></tr>
</table>

The text entered into the text area by using APPEND, READ, or TAPE is always appended to any existing text; if the text area is empty, the text is simply brought into the text area.

Example

```
* /
THIS TEXT IS IN
THE TEXT AREA.
* APPEND ↵
THIS TEXT IS ↵
APPENDED. ↵
DC
* /
THIS TEXT IS IN
THE TEXT AREA.
THIS TEXT IS
APPENDED.
*
```

APPEND

The command

APPEND ↵

is used to enter text from the terminal. After typing the Carriage Return, the user types as many lines of text as desired. The command is terminated with a Control D, which must immediately follow a Carriage Return. If D^C is used anywhere else it will perform its editing function, instead of terminating the APPEND.

Example

```
* APPEND ↵
THIS IS A LINE ↵
THISDC IS A LINE
DC
* /
THIS IS A LINE
THIS IS A LINE
*
```

This D^C copies the rest of the previous line, while the final D^C terminates the APPEND.

NOTE: Many control characters are available for editing text during APPEND. See Use Of Control Characters During APPEND, INSERT, And CHANGE, Page 30.

TAPE

To enter text into EDITOR's text area from paper tape, the command

1 - *a* denotes the address of single line throughout this manual.

TAPE ↵

is used. After typing the Carriage Return, the user turns on the paper tape reader. As the text on the tape is read, the text is printed on the terminal. When the tape has been read, the user turns off the paper tape reader and types a Control D unless there is a D^C prepunched on the tape.

Example

*** TAPE** ↵ *After returning the carriage, the user turns*
26,4.3 *on the paper tape reader.*
27,3.7
35,9.0
36,4.4
40,5.1
33,1.6
35,2.0
46,3.0
54,3.2
40,2.2
47,28
D^C *The user turns off the paper tape reader,*
***** *then types D^C.*

No editing of text is allowed while the tape is being read. However, the control characters

A^C
Q^C
W^C
I^C

will perform their usual editing functions if they are **prepunched** on the tape being read.[1]

READ and *a*READ

Both commands for entering text from a disk file have a long form and a short form. These forms are interchangeable.

To enter text from a file, the user may type

READ ↵

After the Carriage Return is typed, EDITOR responds with

FROM:

The user now types the name of the file and then types a Carriage Return. EDITOR reads in the text and prints the number of words in the file. (3 characters = 1 word).

Example

*** READ** ↵
FROM: /DATA/ ↵
130 WORDS.
*****

The short form of the READ command is

READ file name ↵

If this form were used, the previous example would look like this:

*** READ /DATA/** ↵
130 WORDS.
*****

The above forms of READ simply append the text in the file to any existing text. To read text from a file and insert it before any line already in the text area, precede the READ command by the address of the line **before** which the text is to be inserted.

Example

*** /**
THIS TEXT IS IN
THE TEXT AREA.
*** 'IN'READ** ↵
FROM: /TEXT/ ↵
11 WORDS.
*** /**
THIS TEXT IS READ *These two lines were in the*
AND INSERTED. *file /TEXT/.*
THIS TEXT IS IN
THE TEXT AREA.
*****

NOTE: The maximum amount of text that may be read with the form aREAD is 512 lines. If the user attempts to read more than 512 lines, the error message

TOO MANY LINES

will be printed.

With all forms of the READ command, EDITOR will type a question mark if the file is not found in the user's directory.[2]

No editing can be done while the text is being read. The user must wait for EDITOR to print the asterisk before he begins editing.

Using any form of READ does not erase the file being read.

1 - See *Terminal Input Editing*, Page 23, for the functions of these control characters.

2 - A file in another user's directory may be read if it has an @ in its name. For example, the file /@P/, in the directory with user name JONES and account number A4, may be read using the command READ (A4JONES)/@P/ ↵
Also, a public library program may be read using the command READ *UNISTA ↵

ORGANIZATION OF TEXT IN TEXT AREA

Whether using APPEND, READ, or TAPE, any characters that can be typed on the Tymshare terminal will be accepted as text by EDITOR. However, this text must be organized according to certain principles which are discussed below.

DEFINITION OF A LINE

Text in the text area is divided into lines. EDITOR defines a line of text as any string of characters terminated by a Carriage Return. *NOTE: A Carriage Return alone is considered to be a line.*

Example

```
* APPEND ↵
THIS IS A LINE ↵
SO IS THIS ↵
↵
D^C
* 1/
THIS IS A LINE
* 2/
SO IS THIS
* 3/
*
```

Line 3 consists of a Carriage Return alone.

LINE CONTINUATION: THE LINE FEED

A line of text may consist of more than one physical line as it appears on the terminal. (On most terminals, the maximum physical line is 72 characters.) To enter a line consisting of more than one physical line, end each physical line with a Line Feed.[1]

Example

```
* APPEND ↵
THIS IS ↓
LINE ONE. ↵
THIS IS ↓
LINE ↓
TWO. ↵
D^C
* 1/
THIS IS
LINE ONE.
* 2/
THIS IS
LINE
TWO.
*
```

LINE LENGTH

The maximum allowable line length is 256 characters, including blanks, Line Feeds, and the Carriage Return. If the user attempts to enter a line containing too many characters, the error message

LINE TOO LONG

will be printed. EDITOR will cut off the line by supplying a Carriage Return.

SIZE OF TEXT AREA

EDITOR's large text area may contain up to 60,000 characters. If the user attempts to enter more than this maximum number of characters, whether with APPEND, TAPE, or READ, the error message

TOO MUCH TEXT

will be printed. The command used will be terminated, and text entered up to that point will be retained in the text area.

CAUTION: Since text read into EDITOR from paper tape is also printed at the terminal, if too much text is entered from paper tape the extra text will still be typed on the terminal after termination of the TAPE command. This means that EDITOR will try to interpret the extra text as commands; the most likely result is a series of question marks indicating that EDITOR does not recognize the "commands".

LINE ADDRESSING

Line addresses are used with many EDITOR commands to specify the line or lines upon which the command is to operate. There are four basic kinds of line addresses; these may be used to build more complex addresses as will be shown below.

To illustrate each type of line address, we assume that the following text, a data file containing payroll information, is in the text area:

1 - See Appendix C, *The Terminal*, Page 66, for special conventions concerning Line Feeds and Carriage Returns when punching paper tape off line.

```
ADAMS, JOHN
     1.50 35.50 53.25
BENTLEY, DICK
     2.75 40.00 110.00
BROWN, JANE
     3.00 40.00 120.00
BROWN, FRED
     1.75 38.25 66.94
MEADOWS, BEY
     3.50 40.00 140.00
SWAN, SIGMUND
     2.00 35.25 70.50
UNDERWOOD, SAM
     3.00 40.00 120.00
END
```

BASIC LINE ADDRESSES

1. The EDITOR Line Number

EDITOR assigns a line number to each line of text in the text area; these are consecutive integers beginning with 1.

Example

```
* 2/
     1.50 35.50 53.25
*
```

2. Any Text Within A Line

Any line may be addressed by any text within the line, which must be surrounded by double quotes, single quotes, or square brackets,[1] at the user's option.

Example 1

```
* 'SIG'/
SWAN, SIGMUND
*
```

This line could also be addressed by any of the following:

```
"SIG"
[SIG]
[AN, S]
```

Example 2

```
* [1.50] /
     1.50 35.50 53.25
```

If the text to be used as an address contains any of the characters ', ", [, or], the address may not be surrounded with that character. For example, if the line

```
"JOHNSON" 1.50
```

were in EDITOR's text area, it could be addressed by

```
'"JOHNSON"'
```

or by

```
["JOHN]
```

but **not** by

```
""JOHNSON"
```

It could, of course, be addressed by "JOHNSON".

3. Line Labels

A line label is defined as a string of characters beginning with the first non-blank character in the line, and ending with a non-blank character having a space or Carriage Return immediately after it. A line label may not contain more than one consecutive blank.

Any line may be addressed by a line label, surrounded by colons or angle brackets according to the following rules:

- If the first non-blank character in the line is in print position 1, the line label may be surrounded by colons.

  Example

  ```
  * :BENTLEY,:/
  BENTLEY, DICK
  * :MEADOWS, BEY:/
  MEADOWS, BEY
  * :END:/
  END
  *
  ```

  Two consecutive blanks in this label would be illegal.

- If the line label is preceded by blanks, it **must** be surrounded by angle brackets (< and >). These may also be used to surround labels not preceded by blanks, in place of colons.

  Example

  ```
  * <1.75>/
       1.75 38.25 66.94
  * <3.50 40.00>/
       3.50 40.00 140.00
  * <END>/
  END
  *
  ```

The line label address is a faster means of addressing than text within the line since EDITOR is not required to search the entire line for the specified text.

NOTE: A colon-surrounded line label may not contain a colon; an angle-bracketed line label may not contain an angle bracket.

1 - On the terminal, [is a shift K, and] is a shift M.

4. Special Line Addresses: . And $

The last line in the text area may be addressed by a dollar sign.

```
* $/
END
*
```

The period (.) is used to address the current line, the line which has been operated on most recently. EDITOR keeps track of the current line and redefines it with each use of a command. After the / command, the current line is the last line printed. Thus, since we just printed the last line in the text area, this line may be addressed by . as follows:

```
* ./
END
*
```

The position of the current line after each command introduced thus far in this manual is given in the following table:[1]

Immediately After:	The Current Line (Addressed By .) Is:
APPEND	The last line appended (=$).
DELETE	Line preceding first line deleted.
EDIT	The last line edited.
READ *a*READ	The last line read (=$). The last line of text (=$).
SUBSTITUTE	The last line in which substitutions were made.
TAPE	The last line read from the tape (=$).
WRITE	The last line written on the file.
/	The last line printed.

The current line may be changed by typing a line address followed by a Carriage Return when EDITOR is awaiting a command.

Example

```
* 3↵
* ./
BENTLEY, DICK      This is line 3 in the text area.
```

The location of the current line determines where EDITOR begins searching for a text or line label address; specifically, the search begins at the line following the current line. For example, since there are two lines with label BROWN, in our sample text, the following may occur:

```
* 1/
ADAMS, JOHN        This is now the current line.
* :BROWN,:/
BROWN, JANE
* :BROWN,:/
BROWN, FRED
*
```

Since EDITOR begins searching for the line label BROWN, at the line following the current line, it first prints the line BROWN, JANE. At this point, BROWN, JANE becomes the current line. Thus, typing :BROWN,:/ again causes EDITOR to print the line BROWN, FRED, since it starts searching for the label at the line following BROWN, JANE. *NOTE: If EDITOR reaches the end of the text and still has not found a text or line label address, it will go back to the beginning of the text and continue its search until it either finds the address or reaches the point where it started searching. Thus,*

```
* ./
BROWN, FRED
* :BROWN,:/
BROWN, JANE
*
```

COMBINATION LINE ADDRESSES

Combination line addresses may be formed by concatenating basic addresses. They are useful when a single basic address does not uniquely define a line. For example, the line

BROWN, FRED

in the sample text may be addressed uniquely by "JA":BROWN,: as follows:

```
* "JA":BROWN,:/
BROWN, FRED
*
```

When EDITOR encounters this combination address, it first searches for a line containing JA. It begins its search for this line at the line following the current line. After a line containing JA is found, EDITOR searches for the next line **after** it which contains the line label BROWN,. A look at our sample text shows that this must be the line BROWN, FRED no matter where the search for JA begins, since only one line contains JA:

1 - See Appendix B, Page 65, for a complete table of commands and their effects on the current line.

```
* /
ADAMS, JOHN
     1.50 35.50 53.25
BENTLEY, DICK
     2.75 40.00 110.00
BROWN, JANE
     3.00 40.00 120.00
BROWN, FRED
     1.75 38.25 66.94
MEADOWS, BEY
     3.50 40.00 140.00
SWAN, SIGMUND
     2.00 35.25 70.50
UNDERWOOD, SAM
     3.00 40.00 120.00
END
*
```

EDITOR first finds this line, containing the text JA. This is the next line after the line containing JA which contains the label BROWN,.

The general form of a combination address is

$a_1 a_2 \ldots a_n$

where each a_i is a basic address. The address a_1 may be any basic address (line number, text, label, $, or .). However, a_2 through a_n must be text addresses or line labels. The only limit on the number of addresses that may be concatenated is that the addresses plus the command may not contain more than 256 characters.

When it encounters any combination address, EDITOR first finds the line addressed by a_1. It then searches for the first line after line a_1 that is addressed by a_2. It continues searching thus until a line addressed by a_n is found.

As we have seen, if EDITOR reaches the end of the text and still has not found a text or line label address, it will go back to the beginning of the text and continue its search. Thus, the line

BROWN, JANE

could be addressed by

<2.00>:BROWN,:

To cause EDITOR to begin searching for a text or line label address at line 1, rather than at the line following the current line, a combination address beginning with $ may be used.

Example

```
* $'40.00'/
     2.75 40.00 1.10
*
```

Here, the first line after $ containing the text 40.00 is printed.

ADDRESS ARITHMETIC

Two arithmetic operators, + and -, may be used with a single address and an integer to specify a new address. The general form of such an address is

a + n or a - n

where **a** is any address except an EDITOR line number and **n** is an integer.

The meaning of such addresses is as follows:

a + n Refers to the *n*th line after the line addressed by *a*.

a - n Refers to the *n*th line before the line addressed by *a*.

Examples

```
* "SIG"+2/
UNDERWOOD, SAM

* .+1EDIT ↵
     3.00 40.00 120.00
     4.00 40.00 160.00 ↵

* $-1DELETE ↵
*
```

This command deletes the second to the last line of text.

ADDRESSING A RANGE OF LINES

Most EDITOR commands may be used with the address of a range of lines to specify that the command be executed for all lines in that range.

The address of a range of lines takes the form

a_i,a_f

where a_i is the address of the initial line in the range and a_f is the address of the final line in the range.

Examples

3,[SWAN] / *Prints line 3 through the line containing the text SWAN.*

5,$DELETE ↵ *Deletes line 5 through the last line of text.*

The first address in the range must precede the second address. Thus, for example, 4,1/ is illegal.

NOTE: If the first address in the range is a text or line label address, the current line is reset internally to the first line of the range when that address is found. Thus, the range consisting of the two lines

BENTLEY, DICK
2.75 40.00 110.00

could be addressed by

'DICK',.+1

since the current line is reset to the line containing DICK when the first address in the range is found.

This fact is handy, since it saves the user from typing the text or line label address twice in specifying such ranges. (It is much easier to type 'DICK',.+1 than 'DICK','DICK'+1.) However, caution is necessary. If the first address has the form

a±n

where **a** *is a text or line label address, and* **n** *is an integer, the current line will be reset to the line addressed by* **a** *rather than the line addressed by* **a ± n**.

Example

*** 'BEY'+2,.+5/**	
SWAN, SIGMUND	*This is line 'BEY'+2*
2.00 35.25 70.50	
UNDERWOOD, SAM	
3.00 40.00 120.00	*The current line was reset to the line containing 'BEY'; thus, the last line printed here is the line 5 lines after the line containing BEY.*
*	

OUTPUT OF TEXT FROM TEXT AREA

Text in the text area may be printed at the terminal, punched on paper tape, or written on a disk file. The following output commands are available to the EDITOR user:

<table>
<tr><th>Type Of Output</th><th colspan="2">General Form Of Command</th><th>Command Function</th></tr>
<tr><td rowspan="6">Terminal Output</td><td colspan="2">r/[1]</td><td>Prints the line or lines addressed by r.</td></tr>
<tr><td colspan="2">/</td><td>Prints the entire contents of the text area.</td></tr>
<tr><td colspan="2">rPRINT ↵</td><td>Prints the line or lines addressed by r in a page format.</td></tr>
<tr><td colspan="2">PRINT ↵</td><td>Prints entire contents of text area in a page format.</td></tr>
<tr><td colspan="2">↓</td><td>Prints the line addressed by .+1.</td></tr>
<tr><td colspan="2">↑</td><td>Prints the line addressed by .-1.</td></tr>
<tr><td rowspan="3">Paper Tape Output</td><td colspan="2">rPUNCH ↵ or rPUNCH ↓</td><td>Punches the line or lines addressed by r.</td></tr>
<tr><td colspan="2">PUNCH ↵ or PUNCH ↓</td><td>Punches entire contents of text area.</td></tr>
<tr><td colspan="3">If the Carriage Return is used, EDITOR prints instructions and punches a D[C] at the end of the text. If the Line Feed is used EDITOR does not do either of these.</td></tr>
<tr><td rowspan="4">Disk File Output</td><td>Long Form</td><td>Short Form</td><td rowspan="2"></td></tr>
<tr><td>rWRITE ↵
TO file name ↵</td><td>rWRITE file name ↵</td><td>Writes the line or lines addressed by r to a file. Multiple blanks are compressed.</td></tr>
<tr><td>WRITE ↵
TO file name ↵</td><td>WRITE file name ↵</td><td>Writes entire contents of text area to a file. Multiple blanks are compressed.</td></tr>
<tr><td colspan="2">rWRITE ↓
TO file name ↵

WRITE ↓
TO file name ↵</td><td>Same as rWRITE↵ except multiple blanks are not compressed.

Same as WRITE↵ except multiple blanks are not compressed.</td></tr>
</table>

1 - *r* denotes the address of a single line or a range of lines throughout this manual, unless otherwise specified.

TERMINAL OUTPUT

The / Command

This command has two forms:

/ Prints the entire contents of the text area on the terminal.

r/ Prints the line or lines addressed by *r* on the terminal.

Neither form of this command requires a Carriage Return after it. When the user types the /, EDITOR returns the carriage and prints the specified text.

After the / command, the current line, addressed by ., is defined as the last line printed. Note that if ALT MODE or ESCAPE is used to interrupt the command, the current line is not changed.

Example

```
* /
THIS IS ALL
THE TEXT IN
THE TEXT AREA.
* ./
THE TEXT AREA.
```

The PRINT Command

This command also prints text at the terminal. The difference between it and the / command is that the text is printed in a page format.

Two forms are available:

PRINT ↵ Prints the entire contents of the text area.

*r***PRINT** ↵ Prints the line or lines addressed by *r*.

After the Carriage Return is typed, EDITOR responds with

DOUBLE SPACE?

This question may be answered with a Y (for Yes, print the text with double spacing), or an N (for No, print the text with single spacing).

After the user answers the question, EDITOR returns the carriage and prints the text in the PRINT format.

The PRINT Format

The format used by the PRINT command is designed so that the printout may easily be cut, have holes punched in it, and be placed in a notebook. As described above, the user may specify single or double spacing. He may also specify the number of lines per page, using the LINES command described below.

If the LINES command has not been given, the PRINT command prints 56 lines on an 11 inch page, assuming single spacing. The following format is used:

- After the Y or N response to the question DOUBLE SPACE?, EDITOR gives a series of Line Feeds to separate the command from the printed text.
- EDITOR then types a line with four dashes,

  – – – –

  to serve as a guide for the paper cutter.
- More Line Feeds are given to serve as the upper margin of the page.
- The lines of text are printed.
- More Line Feeds are given, to form the lower margin of the page.
- Another line of four dashes is printed. If another page is needed, it is printed in the same format. Otherwise, EDITOR gives more Line Feeds and then prints the asterisk.

After the PRINT command, the current line, addressed by ., is the last line printed.

The LINES Command

This command sets the number of lines per page for the PRINT command. The general form of the command is

LINES *n* ↵

where *n* is the number of lines per page.

If the command is not given, EDITOR assumes 56 lines per page for PRINT.

Once the command LINES *n* has been given, EDITOR assumes *n* lines per page every time PRINT is used, until another LINES command is given. If the user leaves EDITOR and returns by using CONTINUE, the last LINES command is still in effect since all information is retained when CONTINUE is used. The last LINES command given is also in effect after the CLEAR command.[1]

The LINES command has no effect on the current line.

1 - See Section 4, *UTILITY COMMANDS*, Page 47, for a complete description of CLEAR.

Example: LINES and PRINT

```
* LINES 3↵
* PRINT ↵
DOUBLE SPACE? N

- - - -

A QUICK BROWN
FOX JUMPS OVER
THE LAZY DOG.

- - - -

PICKING JUST SIX QUINCES, NEW
FARMHAND PROVES STRONG BUT LAZY.

- - - -

*
```

The Line Feed And ↑ Commands

If a Line Feed is given when EDITOR is awaiting a command, it prints the next line of text, that is, the line addressed by .+1.

Example

```
* /
THIS IS LINE 1
THIS IS LINE 2
THIS IS LINE 3
* 1/
THIS IS LINE 1
* ↴                    No Carriage Return is re-
THIS IS LINE 2         quired after the ↴
*
```

Note that if the current line is the last line of text, the Line Feed command causes a ? to print. It will not go back to the beginning of the text and print the first line.

Similarly, the ↑ command prints the previous line, addressed by .-1.

Example

```
* ./
THIS IS LINE 2
* ↑                    No Carriage Return is re-
THIS IS LINE 1         quired after the ↑, either.
*
```

If the current line is the first line of text, the ↑ causes a ? to print. After execution of either command, the current line is the line printed.

PAPER TAPE OUTPUT: THE PUNCH COMMAND

This command punches text in the text area on paper tape.

PUNCH may be used with or without the address of a line or a range of lines:

PUNCH Punches the entire contents of the text area.

*r*PUNCH Punches the line or lines addressed by *r*.

In either case, the text is printed at the terminal as well as punched on paper tape.

EDITOR punches header and trailer tape, 3 inches of each, when this command is used.

Two options are available:

Option 1. PUNCH ↵ or *r*PUNCH ↵

If the command is followed by a Carriage Return, EDITOR prints the instructions

```
TURN PUNCH ON, TYPE CONTROL D.
WHEN FINISHED, TURN PUNCH OFF, TYPE
                        CONTROL D.
```

After the user turns on the paper tape punch and types a Control D, EDITOR punches the header followed by the specified text. It then punches a Control D, and following that, the trailer. After EDITOR stops punching tape, the user must turn off the paper tape punch and then type a Control D to terminate the command.

If the tape is subsequently read into EDITOR with the TAPE command, the Control D on the end of the tape will terminate the TAPE command.

Example

```
*:THIS:,:ON:PUNCH ↵
TURN PUNCH ON, TYPE CONTROL D.
WHEN FINISHED, TURN PUNCH OFF, TYPE
                          CONTROL D.
D^C     After turning on the paper tape punch, the
        user types a D^C.
THIS TEXT IS
PUNCHED ON
PAPER TAPE
AND PRINTED
ON THE TERMINAL
D^C     The user turns off the punch, then types D^C.
*
```

Option 2. PUNCH ↓ or *r*PUNCH ↓

If PUNCH is followed by a Line Feed, no instructions are given. EDITOR waits for the user to turn on the paper tape punch and type a Control D. EDITOR then punches the header, text, and trailer. It does **not** punch a Control D at the end of the text. After the trailer is punched, EDITOR waits for the user to turn off the paper tape punch and then type a Control D.

After all variations of PUNCH, the current line, addressed by ., is the last line punched.

FILE OUTPUT: THE WRITE COMMAND

The WRITE command is used to write text on a disk file. It may be used with or without the address of a line or range of lines:

WRITE Writes the entire contents of the text area on a file.

***r*WRITE** Writes the line or lines addressed by *r* on a file.

There are two options:

Option 1. WRITE ↵ or *r*WRITE ↵: Multiple Blank Compression

When the WRITE command is followed by a Carriage Return, the text is written on a file with multiple blanks compressed. This means that EDITOR abbreviates any string of multiple blanks, so that the file will use less space on the disk. The Tymshare languages, CAL, SUPER BASIC, etc., will accept programs written on files with the WRITE↵ command.

This command has a long form and a short form, like the READ command. The long form works as follows:

After the Carriage Return is typed, EDITOR responds with

TO

The user then types the file name followed by a Carriage Return. (Rules for naming files are discussed on Page 21.)

If the user has not previously created a file with this name, EDITOR responds with

NEW FILE

If the user has a file with the chosen name in his directory, EDITOR responds with

OLD FILE

In either case, the user may confirm the command by typing a Carriage Return, or abort it by typing ALT MODE or ESCAPE. If he confirms the command, EDITOR will print the number of words written on the file.

Example

```
*WRITE ↵
TO /PAY/ ↵
 OLD FILE ⊕      The command is aborted. No
*WRITE ↵         new text is written on /PAY/.
TO /NPAY/ ↵
 NEW FILE ↵      This command is confirmed.
252 WORDS.       There are 252 words in the file
*                /NPAY/.
```

NOTE: If a Carriage Return is typed after OLD FILE, whatever was previously on the file will be replaced by the new text.

A short form of the WRITE↵ command is available. The user may just type

WRITE *file name* ↵

Example

```
*WRITE /Q/ ↵
NEW FILE ↵
73 WORDS.
*
```

Option 2. WRITE ↓ or *r*WRITE ↓: No Blank Compression

Although the Tymshare languages handle programs written on files with multiple blanks compressed, command files will not always work properly if they are written with multiple blanks compressed. The WRITE command should be followed by a Line Feed when writing command files in EDITOR. This form of WRITE writes files with blanks uncompressed.

Example

```
* APPEND ↵
SBASIC ↵        The user creates a command file in
                EDITOR.
LOAD /PROG/ ↵
RUN ↵
PRINT "     END OF JOB" ↵
QUIT ↵
FI ↵
D^C
* WRITE ↵       When WRITE↵ is used, EDITOR
TO /COMSB/ ↵    does not print the number of words
 NEW FILE ↵     in the file.
*
```

NOTE: The WRITE↵ command does not have a short form.

After all forms of the WRITE command, the current line, addressed by ., is the last line written on the file.

Using any form of WRITE does not erase the text from the text area.

RULES FOR NAMING FILES

EDITOR will accept all the file names allowed by the Tymshare EXECUTIVE.

1. A file name may begin and end with a slash. Inside the slashes it may contain any characters except a slash.

Examples

```
* READ /@Z1/ ↵
* WRITE /G^C;/ ↵
```

2. A file name may begin and end with single quotes. Inside the quotes, it may contain any characters except a single quote. For example,

```
* WRITE ↵
TO '@ PROGRAM 4' ↵
```

3. File names need not be protected by slashes or quotes. Files which are not thus protected may contain **only** the following characters:

   A through Z
   0 through 9
   @

Example

```
* WRITE ↵
TO @DATA2 ↵
```

SECTION 3
EDITING TEXT IN TEXT AREA

EDITOR has three kinds of commands available for editing text in the text area:

Editing Text Within A Line	Manipulating Whole Lines Or Groups Of Lines	Repetitive Editing
EDIT MODIFY	COPY CHANGE DELETE INSERT MOVE	FIND SUBSTITUTE

In addition, there are 25 editing control characters A^C - K^C and M^C - Z^C which may be used for editing during many commands as well as for editing terminal input. These control characters perform such functions as copying from one line to another, deleting characters in a line, and inserting characters into a line.

TERMINAL INPUT EDITING

Whenever characters are being typed into EDITOR from the terminal, the control characters A^C, Q^C, W^C, and I^C may be used. Thus, these control characters may be used in any of the following situations:

1. When typing commands.

Example

* APPEMA^C←ND ↵ *The A^C deletes the preceding character, M.*

NOTE: There is one exception to this rule; file names may not be edited. Thus, if an error is made while typing the file name during READ or WRITE, press the ALT MODE or ESCAPE key and retype the entire command.

2. When typing text during APPEND, INSERT, and CHANGE.[1]

Example

* APPEND ↵
FOURCORE AND SEVEN YEARS AGOQ^C↑
The Q^C deletes the entire line being typed and returns the carriage; the user then types the correct line.
FOURSCORE AND SEVEN YEARS AGO ↵

3. When typing the new line during EDIT and MODIFY.

Example

* 2 EDIT ↵
OUR FATHERS FORTH
OUR FATHERS BOUGHTW^C\BROUGHT FORTH↵
The W^C deletes the preceding word in the line being typed.
* ./
OUR FATHERS BROUGHT FORTH
*

4. When using the TAPE command.

Although no editing may be done while the tape is being read, A^C, Q^C, W^C, and I^C will be accepted by EDITOR's TAPE command if they have been **prepunched** on the tape being read.

CONTROL A: DELETING THE PRECEDING CHARACTER

This control character deletes the immediately preceding character; EDITOR prints a back arrow (←) to indicate its use. It may be used repeatedly to delete

1 - See Page 32 for a description of INSERT and CHANGE. The form of these commands is similar to that of APPEND; the control characters discussed here work the same way during INSERT and CHANGE as they do during APPEND.

more than one preceding character up to and including the first character in the line being typed. It will not delete characters in preceding lines.[1]

Example

```
* APPEND ↵
THIS IX TA^C←A^C←A^C←S TEXT ↵
```

The first A^C deletes the T; the second, the space; the third, the X. The rest of the line is retyped correctly.

```
D^C
* ./
THIS IS TEXT
*
```

CONTROL Q: DELETING THE PRECEDING LINE

Control Q deletes the entire line being typed. EDITOR prints an up arrow (↑) to indicate its use, and returns the carriage for the user.

Example

```
* :12:EDIT ↵
12 24.36 89 32
13 24.19 32Q^C↑
13 34.19 32 ↵
* ./
13 34.19 32
*
```

When typing text during APPEND, INSERT, and CHANGE, Control Q may be used to delete several lines of text, up to and including the first line of text typed during the command.

Example

```
* APPEND ↵
CONTROL Q WILL ↵
DELETE MORE THAN ↵
ONE LINE ↵
DURING APPEND ↵
INSERT, AND ↵
CHANGEQ^C↑
Q^C↑
Q^C↑
D^C
* /
CONTROL Q WILL
DELETE MORE THAN
ONE LINE
*
```

CONTROL W: DELETING THE PRECEDING WORD

This control character deletes the preceding word in the line being typed. The preceding word is defined as including:

- The immediately preceding blanks, if any, plus
- The immediately preceding non-blank characters, up to but **not** including the first blank preceding them.

EDITOR prints a back slash (\) when a Control W is used.

Example

```
  APPEND ↵
ONE    TWOW^C\THREE ↵
TWO    GOUR  W^C\FOUR ↵
D^C
* /
ONE    THREE
TWO    FOUR
*
```

Note that the two spaces following GOUR are deleted, but the three spaces preceding the word are not.

Control W may be used repeatedly to delete more than one word in the line, up to and including the first word in the line. It may not be used to delete words in preceding lines.

CONTROL I: TABS

Control I is used to space to the next tab stop. EDITOR automatically initializes 10 tab stops, at print positions 8, 16, 32, 40, 45, 50, 55, 60, 65, and 70. These may be changed with the TABS command (see Section 4).

Example

```
* APPEND ↵
12345678901234567890123456789012345678901234567890 ↵
I^C      ↑I^C      ↑I^C                    ↑I^C      ↑ ↵
D^C
* /
12345678901234567890123456789012345678901234567890
        ↑         ↑                       ↑         ↑
*
```

In this example, the user appended a line of forty digits. In the next line, he typed I^C; EDITOR responded by spacing to the first tab stop at print position 8. The user typed an up arrow to mark the spot, and continued typing I^C followed by ↑ until he reached the tab stop at print position 40.

1 - It is not usually a good idea to use A^C during EDIT and MODIFY; see *Control N*, Page 29.

EDITING TEXT WITHIN A LINE

THE EDIT AND MODIFY COMMANDS

The EDIT and MODIFY commands are used to make changes in individual lines of text that are in the text area. The following forms of these commands are available:

*r*EDIT ↵	*Allow edit of the line or lines addressed by r.*
*r*MODIFY ↵	
EDIT ↓	*Allow edit of the next line, addressed by .+1.*
MODIFY ↓	
EDIT↑	*Allow edit of the previous line, addressed by .-1.*
MODIFY↑	

Editing A Single Line

When an EDIT or MODIFY command is given with the address of a single line, the line addressed is made available for editing. This line is called the **old line**; it is the line printed on the terminal when the EDIT command is used. MODIFY is the same as EDIT except that it does not print the old line.

After the EDIT or MODIFY command is given (and the old line is typed, with EDIT), the user may type the line which is to replace the old line, called the **new line**, followed by a Carriage Return. The Carriage Return terminates the new line and the EDIT or MODIFY command, and the new line replaces the old line in the text area.

Example

```
* :SMITH,:EDIT ↵
SMITH, JOHN 1.50          EDITOR prints the old line.
SMITH, JOHN 2.00 ↵        The user types the new line.
* ./
SMITH, JOHN 2.00          The new line has replaced
*                         the old line in the text area.
```

The next example shows the same editing done with MODIFY. The only difference is that the old line is not printed.

```
* :SMITH,:MODIFY ↵
SMITH, JOHN 2.00 ↵        This is the new line.
* ./
SMITH, JOHN 2.00
*
```

The forms EDIT↓ and MODIFY↓ are equivalent to .+1 EDIT and .+1 MODIFY, respectively. Similarly, EDIT ↑ and MODIFY ↑ are equivalent to .-1 EDIT and .-1 MODIFY, respectively. It is unnecessary to type a Carriage Return after the Line Feed or up arrow.

Editing A Range Of Lines

If EDIT is used with the address of a range of lines, EDITOR prints the specified lines one at a time. After printing an old line, it waits for the user to edit it, and then prints the next line in the range, continuing thus until all lines in the range have been edited.

Example

```
* /
LINE 4
LINE 5
LINE 6
*1,3EDIT ↵
LINE 4
LINE 1 ↵
LINE 5
LINE 2 ↵
LINE 6
LINE 3 ↵
* /
LINE 1
LINE 2
LINE 3
*
```

MODIFY used with the address of a range of lines works the same way, except that the old lines are not printed.

The current line, addressed by ., is defined as the last line edited after all forms of EDIT and MODIFY. *NOTE: This implies that successive lines of text may be edited by using EDIT↓ or MODIFY↓ repeatedly.*

CONTROL CHARACTERS USED WITH EDIT AND MODIFY

In addition to the control characters A^C, Q^C, W^C, and I^C discussed under *Terminal Input Editing*, there are many more control characters which may be used during EDIT and MODIFY.[1] These characters all perform some sort of editing operation on the old line to form the new line. For example, some of them copy characters from the old line to the new line.

In the following discussion of control characters used with EDIT and MODIFY, the control characters

1 - These additional control characters may also be used during APPEND, INSERT, and CHANGE; see *Use Of Control Characters During APPEND, INSERT, And CHANGE*, Page 30.

are classified according to the functions they perform. Some of these control characters also terminate the edit; this will be noted under the description of the character.

Control Characters Used For Deleting

Control S

Control S deletes the next character in the old line. EDITOR prints a percent sign (%) to indicate that a Control S was used.

Example

```
* 'AMER'EDIT ↵
AMERICAN TEL & TEL      256,960  This is the
                                 old line.
Z^C6AMERICAN TEL & TEL     256S^C%960 ↵
* ./
AMERICAN TEL & TEL      256960   This is the
                                 new line.
*
```

In this example, Control Z copies characters from the old line to the new line up to and including the character typed after it, 6. At this point, the next character in the old line is a comma, which the user wishes to delete. He types a Control S and EDITOR prints a %. The comma is now deleted. The user then types the rest of the new line and terminates the edit with a Carriage Return.

Control K

This control character deletes the next character in the old line and prints the character it deletes. Thus, it performs exactly the same function as Control S; the only difference between Control K and Control S is that EDITOR prints the character deleted instead of printing a %. Thus, Control K could have been used in the above example to delete the comma, as follows:

```
* 'AMER'EDIT ↵
AMERICAN TEL & TEL      256,960
Z^C6AMERICAN TEL & TEL     256K^C,960 ↵
* ./
AMERICAN TEL & TEL      256960
*
```

Control P

Control P deletes characters from the old line up to but **not** including the character typed after it. EDITOR prints a percent sign for each character it deletes.

Example

```
* :STOCKS:EDIT ↵
STOCKS     ASTRODATA      1399
P^CA%%%%%%%%%D^CASTRODATA      1399
* ./
ASTRODATA      1399
*
```

Here the user wishes to delete the word STOCKS as well as the four blanks following it; in other words, he wants the new line to begin with the word ASTRODATA. He types Control P followed by the character A, telling EDITOR to delete the characters up to but not including the first A it encounters. EDITOR prints a % for each character deleted. The user then types a Control D to copy the rest of the old line to the new line and end the edit.

Control X

Control X deletes characters from the old line up to **and** including the character typed after it. Again a % is printed for each character deleted.

Example

```
* EDIT ↵
CANOGA ELECTRONICS      SHARES OUT
                        STANDING=511
Z^CSCANOGA ELECTRONICS     X^C=%%%%%%%
                        %%%%%%%%%%%511 ↵
* ./
CANOGA ELECTRONICS      511
*
```

In this example, the user types Z^CS to copy characters from the old line to the new line up to and including the character S. He then types five blanks to make sure that the new line contains appropriate spacing, and types a Control X followed by an equal sign to delete the characters up to and including the equal sign. Then he types the rest of the line, terminating the edit with a Carriage Return.

Control Characters Used For Copying

There are 10 control characters used for copying from the old line to the new line. C^C, O^C, Z^C, and U^C simply copy characters; D^C and F^C copy the rest of the old line and end the edit; and H^C, R^C, T^C, and Y^C copy the rest of the old line and continue the edit, either at the end of the old line (H^C) or at some point in the new line (R^C, T^C, and Y^C).

Control C

Control C copies the next character in the old line; it may be used repeatedly to copy more than one character.

Example

```
* 8EDIT ↵
COLINS RADIO      2899
C^CC^OE^<LE^>D^LINS RADIO      2899
*./
COLLINS RADIO     2899
*
```

In this example, the user wishes to correct a spelling error, where an L was left out of COLLINS. He copies the C and O using Control C twice and then uses Control E to insert an L. A Control D copies the rest of the old line and ends the edit.

Control O

Like Control X, P, and Z, Control O may be followed by any character. It copies up to but **not** including the character typed after it.

Example

```
* 'STOCK'EDIT ↵
CONDUCTRON STOCKS      2536
O^ CONDUCTRONX^K%%%%%%S^%D^      2536
*./
CONDUCTRON      2536
*
```

In the above edit, the user deletes the word STOCKS (and the space preceding it) from the old line. He first copies up to but not including the first space in the line by typing Control O followed by a space. He then deletes the characters up to and including the K with X^K, and deletes the final S in STOCKS with Control S. The edit is completed with a Control D.

Control Z

Control Z copies up to **and** including the character typed after it. The line edited in the previous example could also be edited using Control Z as follows:

```
* 'STOCK'EDIT ↵
CONDUCTRON STOCKS      2536
Z^ CONDUCTRON X^ %%%%%%%D^      2536
*./
CONDUCTRON      2536
*
```

Since he types a Control Z followed by a space, the user has copied the first word in the line plus the space after it. Since he wishes to preserve five spaces between the name of the company and the number following it, he uses X^ followed by a space to delete the word STOCKS and the space immediately following it. The D^ completes the edit.

Control U

This control character copies characters from the old line to the new line up to but not including the character at the next tab stop.[1]

Example

```
* .EDIT ↵
1234567890
U^1234567 ↵
*./
1234567
*
```

Since the first tab stop is at print position 8, the Control U in this example copies the first 7 characters in the line.

NOTE: If the next tab stop is at a print position beyond the last character in the old line, Control U effectively ends the edit since it copies the Carriage Return. Thus, if another Control U were typed in the above example, the following would occur:

```
*.EDIT ↵
1234567890
U^1234567U^890
*
```

Control D

Control D copies the rest of the old line to the new line, printing it, and ends the edit.

Example

```
* :CORP.:EDIT ↵
CORP.      4210
E^<DYNAMICS E^>D^CORP.      4210
*./
DYNAMICS CORP.      4210
*
```

Control F

Like Control D, Control F copies the rest of the old line to the new line and ends the edit; however, it does not print the characters copied. Thus, using Control F instead of Control D in the above example would yield the following:

1 - Unless the TABS command (see Page 47) has been given, EDITOR assumes tab stops at print positions 8, 16, 32, 40, 45, 50, 55, 60, 65, and 70.

```
* :CORP.:EDIT ↵
CORP.      4210
E^C<DYNAMICS E^C>F^C
* ./
DYNAMICS CORP.      4210
*
```

Control H

Control H copies the rest of the old line to the new line and prints it. It does **not** end the edit; thus, editing may continue at the end of the line.

Example

```
* [SPECIAL]MODIFY ↵
H^CELECTRONIC SPECIALTY      1648 ↵
* ./
ELECTRONIC SPECIALTY      1648
*
```

In this example, the MODIFY command is used instead of the EDIT command, so that the old line is not printed. The H^C copies the rest of the old line, which happens to be the entire old line in this case. The edit continues at the end of the line, where the user adds an 8 to the line and terminates the edit with a Carriage Return.

Control R

Control R gives a Line Feed without returning the carriage, copies the rest of the old line, and then returns the carriage and copies the new line up to the point where the Control R was typed. The edit continues at this point in the new line. Thus, Control R is especially useful when a great deal of editing has been done and the user wishes to see what the new line looks like thus far.

Example

```
* <LINCH>EDIT ↵
     LINCH SYSTEMS      722
P^CL%%%%LTN^C←UN^C←YNCHR^C
                        SYSTEMS      722
LYNCH
```

The first editing the user wishes to do in the above example is to delete the four spaces preceding the word LINCH, and change the I in LINCH to a Y. The P^CL deletes the four spaces. The user then types an L, but makes two typing errors when trying to type a Y. He uses Control N to backspace over these errors (see Control N below). After he finally types the YNCH, he decides to use Control R to see exactly what he has done to the new line. The R^C gives a Line Feed, types the rest of the old line, gives a Carriage Return, and types the new line up to the point where the R^C was typed. The user is positioned at the end of the word LYNCH and may continue the edit from this point.

Note that the Control R does not align the old and new lines. The following control character, Control T, will do this, although it is thus a little slower than Control R.

Control T

Control T is identical to Control R, except that it aligns the old and new lines, thus typing them in a more readable format. Control T copies the rest of the old line plus the new line up to the point where the Control T was typed, aligning the old and new lines, and continues the edit at this point. Thus, using Control T instead of Control R in the above would yield the following:

```
* <LINCH>EDIT ↵
     LINCH SYSTEMS      722
P^CL%%%%LTN^C←UN^C←YNCHT^C
        SYSTEMS      722
LYNCH
```

The user may now continue editing with the rest of the old line printed directly above the point at which he types the new line.

Control Y

Control Y copies but does not print the rest of the old line, and continues the edit at the beginning of the new line, with the new line used as the old line. It is useful when the user has done some editing in the line, and wishes to make a change earlier in the line without losing the editing he has done already, as he would if he used Control Q to restart the edit.

Example

```
* 'WATKINS'EDIT ↵
WATKINS JOHNSON      2731
Z^C2WATKINS JOHNSON      26Y^C
Z^CSWATKINS–F^C
* ./
WATKINS–JOHNSON      2631
*
```

The user copies up to and including the 2 with Z^C2; then types a 6 to replace the 7 in the old line. At this point, he realizes that he has forgotten to type a dash between WATKINS and JOHNSON, so he uses Control Y to copy the rest of the old line and continue the edit at the beginning of the new line. Note that the rest of the old line does not print on the terminal. The user now copies out to the S with Z^CS, types a –,

and copies the rest of the old line and ends the edit with F^C. Note that F^C copies the edited line rather than the original old line.

Inserting Characters Into A Line: Control E

Control E inserts text into the old line. When text is to be inserted, a Control E is typed first; EDITOR prints a <. Then the text to be inserted is typed; this is followed by a second E^C. This time, EDITOR prints a >.

Example

```
*'WESTERN'EDIT ↵
WESTERNUNION      7527
Z^CNWESTERNE^C< E^C>F^C
*./
WESTERN UNION     7527
*
```

In this example, Control E is used to insert a space between the words WESTERN and UNION.

Other Control Characters

Control N: Backspacing

Control N backspaces one character in both the old and the new lines. EDITOR prints a back arrow (←) when Control N is used.

```
*.EDIT ↵
BBCDEF
NOPN^C←N^C←N^C←AD^CBCDEF
*./
ABCDEF
*
```

In this example, the user types the characters NOP in the new line. He then uses Control N three times to backspace over these characters. Since Control N backspaces in the old line as well as the new line, he is now positioned at the beginning of the old line. He types an A, to replace the first B in the old line. Now he is positioned at the second character in the old line. Thus, the rest of the old line consists of the characters BCDEF. Control D copies these characters from the old line to the new line and ends the edit.

▶ *NOTE:* During EDIT and MODIFY, it is usually better to delete characters in the new line by backspacing over them with Control N, rather than using Control A. Since Control A does not backspace in the old line, it is very easy to lose characters accidentally from the old line when using Control A. The following two examples clarify this difference between Control A and Control N:

```
*:DATA:EDIT ↵
DATA TREMD      471
Z^CEDATA TRE,N^C←ND^CD      471
*./
DATA TREND      471
*
```

When attempting to change the M to an N, the user mistakenly types a comma, which replaces the M in the old line. The next character in the old line is now the D. Since he wishes to replace the M rather than the D, he must backspace in the old line and the new line with Control N. If he uses Control A instead, he will lose a character, as follows:

```
*:DATA:EDIT ↵
DATA TREMD      471
Z^CEDATA TRE,A^C←ND^C      471
*./
DATA TREN      471
*
```

Typing a Control A does not change the position in the old line. Thus, when the N is typed, it replaces the D in the old line. Then a Control D copies the rest of the old line, which consists of the characters from the blank following the D through the 1.

Control Q: Restarting The Edit

We have already discussed Control Q under *Terminal Input Editing;* however, we should mention here that when it is used during EDIT and MODIFY, it not only deletes the new line but also restarts the edit at the beginning of the old line. Thus, it is equivalent to using Control N repeatedly to delete the entire new line, rather than using Control A to do this.

NOTE: Control W works like Control A during EDIT and MODIFY in that it does not back up in the old line.

Control M and Control J

Control M and Control J are equivalent to the Carriage Return and Line Feed, respectively. They may be used any time a Carriage Return or Line Feed is desired during any EDITOR command.

The Carriage Return During EDIT And MODIFY

During EDIT and MODIFY, the Carriage Return terminates the new line and the edit.

Example

```
* 1EDIT ↵
LIST OF STOCKS
Z^CTLIST ↵
* ./
LIST
*
```

USE OF CONTROL CHARACTERS DURING APPEND, INSERT, AND CHANGE

All the control characters which may be used during EDIT and MODIFY may also be used during APPEND, INSERT, and CHANGE.[1] In essence, most of them work the same way they do during EDIT and MODIFY. However, because of the way the old and new lines are defined during APPEND, INSERT, and CHANGE, the result of using control characters which operate on the old line is somewhat different during these commands.

During APPEND, INSERT, and CHANGE, the old line is defined as the line immediately preceding the line being typed, while the new line is defined as the line being typed. **This means that BOTH the new and the old lines will be placed in the text area.** The new line does not replace the old line in the text area as it does during EDIT and MODIFY.

Example

```
* APPEND ↵
1,0,0,1,0,0 ↵
F^C
F^C
D^C
* /
1,0,0,1,0,0
1,0,0,1,0,0
1,0,0,1,0,0
*
```

Note that Control F copies the rest of the old line without printing it and terminates the new line, whereas Control D does not copy anything in the above example; it merely terminates the command.

Control D During APPEND, INSERT, And CHANGE

During APPEND, INSERT, and CHANGE, Control D may either terminate the command, as in the above example, or it may copy the rest of the old line to the new line and terminate the new line. Which of these actions it performs depends on where it is typed.

- If it is typed immediately after a Carriage Return, it terminates the command.
- Otherwise, it copies the rest of the old line to the new line, prints it, and terminates the new line with a Carriage Return.

Example

```
* APPEND ↵
1,0,0,0,0 ↵
0,1,D^C0,0,0      The first two D^C's copy the rest of
0,0,1,D^C0,0      the old line to the new line and
                  terminate the new line.
D^C               The final D^C terminates the AP-
* /                PEND.
1,0,0,0,0
0,1,0,0,0
0,0,1,0,0
*
```

Control D is the only control character which has a slightly different function during APPEND, INSERT, and CHANGE; all other control characters perform the same functions as they do during EDIT and MODIFY. As long as the difference in the definitions of old line and new line is kept in mind, there should be no trouble understanding most of the control characters. There are a few; namely, Y^C, R^C, and T^C, whose functions during APPEND, INSERT, and CHANGE are not obvious. These are discussed below.

Control Y During APPEND, INSERT, And CHANGE

During these commands, Control Y copies the rest of the old line to the new line, returns the carriage, and then allows an edit of the new line. Only one line is added to the text being typed, no matter how many times Control Y is used before terminating the new line.

Example

```
* APPEND ↵
1,0,0,2,0,0 ↵
3,1,2,Y^C         The Y^C copies the 2,0,0 and allows an
2D^C,1,2,2,0,0    edit of line 2.
D^C
* /
1,0,0,2,0,0       Note that only two lines are appended.
2,1,2,2,0,0
*
```

To see the difference between Y^C and other control characters that copy the rest of the old line, con-

1 - INSERT and CHANGE are discussed on Page 32. Syntactically, they are similar to APPEND. The control characters discussed here work the same way during INSERT and CHANGE as they do during APPEND.

sider what happens if Control F is used instead of Control Y in the above example:

```
* APPEND ↵
1,0,0,2,0,0 ↵
3,1,2,F^C          The F^C copies the 2,0,0 and ends
2D^C,1,2,2,0,0     the new line.
D^C
* /
1,0,0,2,0,0        Here, three lines are appended.
3,1,2,2,0,0
2,1,2,2,0,0
*
```

CAUTION: When using Control Y to allow editing of the new line, it is very easy to forget that it also copies the rest of the old line. If the old line is longer than the new line, unwanted text may accidentally be added to the text area, as in this example:

```
* APPEND ↵
      500     1.95    40     78.00 ↵
FRAMER, FREDY^C    The user wanted to correct an
FARF^C             error at the beginning of the
D^C                new line. He forgot about the
* /                 old line!
      500     1.95    40     78.00
FARMER, FRED.95    40     78.00
*
```

Control R And T During APPEND, INSERT, And CHANGE

During these commands, Control R copies the rest of the old line plus the new line up to the point where the R^C was typed, and allows the user to continue typing the new line at this point, using control characters if desired. Control T is the same as Control R except that it aligns the old and new lines. Like Control Y, these characters may be used as many times as desired before terminating the new line, and only one new line will be added to the text being typed.

Example

```
* APPEND ↵
HICKORY        DICKORY        DOCK ↵
THE MOUSE TN^C←RAN UP THHN^C←E CLT^C
                                OCK
THE  MOUSE  RAN  UP  THE  CLF^C
D^C
* /
HICKORY        DICKORY        DOCK
THE  MOUSE  RAN  UP  THE  CLOCK
*
```

MANIPULATION OF WHOLE LINES AND GROUPS OF LINES

In addition to editing text within a line, the EDITOR user may delete whole lines of text, insert lines of text, change lines, copy lines, and move lines to a different position in the text. The commands to manipulate whole lines and groups of lines are very easy to remember since they are all descriptive of the functions they perform. They are:

DELETE
INSERT
CHANGE
COPY
MOVE

THE DELETE COMMAND

This command has the general form

*r*DELETE ↵

where *r* is the address of a single line or a range of lines. The command deletes the line or lines addressed by *r*.

Example

```
* /
THESE LINES WILL
BE DELETED.
THESE LINES WILL
REMAIN IN THE TEXT AREA.
* 1,2 DELETE ↵
* /
THESE LINES WILL
REMAIN IN THE TEXT AREA.
*
```

Note that each time the DELETE command is used, EDITOR reassigns line numbers. EDITOR line numbers are **always** consecutive integers beginning with 1; thus, in the above example, lines 3 and 4 become lines 1 and 2 after the original lines numbered 1 and 2 are deleted.

The DELETE command may also be used with a Line Feed to delete the next line. Thus, DELETE↴ is equivalent to .+1DELETE.

After the DELETE command, the current line, addressed by ., is the line preceding the first line deleted. If the first line of text is deleted, the current line is simply the first line of text remaining in the text area.

The DELETE command may be used to delete all lines of text in the text area by typing

1,$DELETE ↵

However, this may also be accomplished with the CLEAR command, see Page 47.

THE INSERT COMMAND

The general form of this command is

*a***INSERT ↵**
text to be
inserted ↵
D^C

where *a* is the address of a single line. The text typed during the command is inserted into the text area **before** the line addressed by *a*. The user may type as many lines of text as desired before terminating the command with a Control D. The terminating **D^C must** follow a Carriage Return.

All of the editing control characters that may be used with APPEND may be used with INSERT; they work the same way during INSERT as they do during APPEND.[1]

Example

```
* /
NOW IS THE TIME
OF THEIR PARTY.
* $INSERT ↵
FOR ALL GOOD MEN ↵
TO CIA^C←OME TO THE AID ↵
D^C
* /
NOW IS THE TIME
FOR ALL GOOD MEN
TO COME TO THE AID
OF THEIR PARTY.
*
```

Note that the text is inserted before the line addressed by $.

EDITOR reassigns line numbers after the INSERT command. Thus, in the above example, the last line of text has line number 2 before the INSERT command, whereas afterwards it has line number 4.

After INSERT, the current line, addressed by ., is the last line typed during the INSERT.

THE CHANGE COMMAND

This command has the general form

*r***CHANGE ↵**
lines of text
to replace the
lines addressed
by *r* ↵
D^C

It is used to replace a line or a range of lines already in the text area (addressed by *r*) with the line or lines typed during the command.

The number of lines in the range *r* need not be the same as the number of lines typed to replace them. Thus, for example, the CHANGE command may be used to replace 1 line with 3 lines, or to replace 11 lines with 4 lines. As many lines as desired may be typed during the command to replace as many lines as desired.

Just like APPEND and INSERT, the CHANGE command is terminated with a Control D, which must immediately follow a Carriage Return.

All of the editing control characters that may be used with APPEND and INSERT may be used with CHANGE; they work the same way during CHANGE as they do during APPEND and INSERT.[1]

Example

```
* /
NOW IS THE TIME FOR
THE LAZY DOG.
* 'TIME'CHANGE ↵
THE QUICK BROWN ↵
FOX JUMPS OVER ↵
D^C
* /
THE QUICK BROWN
FOX JUMPS OVER
THE LAZY DOG.
*
```

These two lines will replace line 1.

EDITOR reassigns line numbers after CHANGE just as after DELETE and INSERT.

The current line after CHANGE is defined as the last line typed during the command.

THE COPY COMMAND

This command has the general form

*a***COPY *r* ↵**

It copies the line or range of lines addressed by *r* and inserts them **before** the line addressed by *a*. Up to

1 - See *Use Of Control Characters During APPEND, INSERT, and CHANGE,* Page 30.

and including 512 lines may be copied with a single command.

Attempting to copy more than 512 lines results in the error message:

```
TOO MANY LINES
```

Example

```
*/
THIS LINE WILL BE COPIED.
THEN THIS RANGE OF
LINES WILL BE COPIED.
*:THEN:COPY 1↵
*/
THIS LINE WILL BE COPIED.
THIS LINE WILL BE COPIED.
THEN THIS RANGE OF
LINES WILL BE COPIED.
*1 COPY:THEN:,:LINES:↵
*/
THEN THIS RANGE OF
LINES WILL BE COPIED.
THIS LINE WILL BE COPIED.
THIS LINE WILL BE COPIED.
THEN THIS RANGE OF
LINES WILL BE COPIED.
*
```

After the COPY command, the current line, addressed by ., is defined to be the first of the lines which are copied and inserted. Thus, immediately after the second COPY command given in the above example, the current line is line 1,

```
THEN THIS RANGE OF
```

THE MOVE COMMAND

The general form of the MOVE command is

*a*MOVE *r*↵

This command moves the line or range of lines addressed by *r* in front of the line addressed by *a*. Up to and including 512 lines may be moved with a single MOVE command.

Attempting to move more than 512 lines results in the error message:

```
TOO MANY LINES
```

Example

```
*/
THIS IS LINE ONE
THIS IS LINE TWO
THIS IS LINE FOUR
THIS IS LINE FIVE
THIS IS LINE THREE
THIS IS LINE SIX
*[SIX]MOVE[FOUR],[FIVE]↵
*/
THIS IS LINE ONE
THIS IS LINE TWO
THIS IS LINE THREE
THIS IS LINE FOUR
THIS IS LINE FIVE
THIS IS LINE SIX
*
```

REPETITIVE EDITING

Thus far, we have discussed commands which enable the EDITOR user to edit text within a line and to manipulate whole lines of text. We come now to two commands which may be used to edit repetitively, SUBSTITUTE and FIND. The SUBSTITUTE command allows mass substitutions of characters throughout any part of the text area. The FIND command is even more powerful. It allows the user to execute other commands repeatedly; for example, it might be used to delete all lines containing the text 'JOB'. It is also useful for information retrieval; for example, it can be used to find all lines containing any text and print them.

In this section we also discuss two more control characters, Control G and Control V, which are especially useful with SUBSTITUTE and FIND.

THE SUBSTITUTE COMMAND

This command may be used in either of the following forms:

SUBSTITUTE↵	*Makes substitutions throughout the text.*
*r***SUBSTITUTE**↵	*Makes substitutions only in the line or lines addressed by* **r.**

Making Substitutions Throughout The Text

To make substitutions throughout the text, the SUBSTITUTE command is used without an address as follows:

```
SUBSTITUTE ↵
"any charactersD^C" FOR "characters in text areaD^C"
WAIT?
```

After the user types SUBSTITUTE↵, EDITOR prints a ". The user then types the characters to be inserted, terminated by a Control D. EDITOR now prompts the user by printing " FOR ". The user types the characters to be replaced, again terminating them with a Control D. EDITOR prints another ", returns the carriage, and asks the question WAIT? The user now has two options:

1. He may tell EDITOR to make all specified substitutions by answering WAIT? with an N (for No). EDITOR will return the carriage, make all substitutions in every line which contains the characters to be replaced, and print the number of substitutions made.

Example

```
* /
ADAMS          $1.50/HOUR
BENTLEY        $2.75/HOUR
BROWN          $3.00/HOUR
DEARBORN       $1.75/HOUR
FIELD          $1.50/HOUR
GREER          $4.75/HOUR
LAMONT         $1.25/HOUR
MEADOWS        $3.50/HOUR
MITTY          $5.50/HOUR
RUFOLO         $3.25/HOUR
SMITH          $1.50/HOUR
SOUTHERN       $2.50/HOUR
* SUBSTITUTE ↵
"HRD^C" FOR "HOURD^C"
WAIT? N        The N causes EDITOR to make all
12             substitutions automatically. EDI-
* /            TOR returns the carriage and prints
               the number of substitutions made,
               12.
ADAMS          $1.50/HR
BENTLEY        $2.75/HR
BROWN          $3.00/HR
DEARBORN       $1.75/HR
FIELD          $1.50/HR
GREER          $4.75/HR
LAMONT         $1.25/HR
MEADOWS        $3.50/HR
MITTY          $5.50/HR
RUFOLO         $3.25/HR
SMITH          $1.50/HR
SOUTHERN       $2.50/HR
*
```

2. He may selectively substitute, telling EDITOR when to make the specified substitution and when not to make it. This is done by typing a Y (for Yes) after the question WAIT?. EDITOR will then print the first line of text that contains the characters to be replaced, and ask the question

   OK?

   A Y typed in answer to this question tells EDITOR to make the substitution for the **first occurrence** of the characters to be replaced in that line. An N tells EDITOR not to make that substitution.

   If there is only one occurrence of the characters to be replaced in the line printed, EDITOR prints the next line and asks OK? again.

Example

```
* APPEND ↵
5 CUPS FLOUR ↵
3 CUPS SUGAR ↵
1 STICK BUTTER ↵
5 CUPS SALT ↵
3 TABLESPOONS WATER ↵
5 CUPS BAKING SODA ↵
D^C
* SUBSTITUTE ↵
"1 TEASPOOND^C" FOR "5 CUPSD^C"
WAIT? Y
5 CUPS FLOUR
OK?N
5 CUPS SALT
OK?Y
5 CUPS BAKING SODA
OK?Y
2              EDITOR has made 2 substitutions out of 3
* /            lines considered.
5 CUPS FLOUR
3 CUPS SUGAR
1 STICK BUTTER
1 TEASPOON SALT
3 TABLESPOONS WATER
1 TEASPOON BAKING SODA
*
```

If a line contains more than one occurrence of the characters to be replaced, the question OK? is asked for each occurrence of the characters in the line, as follows: After the first OK?, EDITOR prints the same line again, this time omitting the first part of the line, up to and including the characters first considered. This enables the user to see which characters EDITOR is asking about. Now OK? is asked for the second occurrence of the characters to be replaced. EDITOR continues in this manner until all occurrences of the characters to be replaced have been considered, and then goes to the next line.

Example

```
* /
10 DATA 1.0 5.1 3.2
20 DATA 4.1 6.2 8.9
* SUBSTITUTE ↵
",DC" FOR " DC"          The user wishes to separate
WAIT? Y                  his data values with commas instead of spaces.
10 DATA 1.0 5.1 3.2
OK?N                     This OK? refers to the first
DATA 1.0 5.1 3.2         space in the line...
OK?N                     This, to the second...
1.0 5.1 3.2
OK?Y                     This, to the third...
5.1 3.2
OK?Y                     And this, to the fourth.
20 DATA 4.1 6.2 8.9      Now, the second line is
OK?N                     examined.
DATA 4.1 6.2 8.9
OK?N
4.1 6.2 8.9
OK?Y
6.2 8.9
OK?Y
4                        EDITOR has made 4 substitutions in 2 lines.
* /
10 DATA 1.0,5.1,3.2
20 DATA 4.1,6.2,8.9
*
```

After EDITOR has thus questioned and received replies for each occurrence of the characters to be replaced in each line, it prints the number of substitutions made.

Making Substitutions In A Specified Range

To make substitutions in a line or lines of text, the form

```
rSUBSTITUTE ↵
"any charactersDC" FOR "characters in text areaDC"
WAIT?
```

is used, where *r* is the address of a line or a range of lines. This form works exactly like SUBSTITUTE without an address, except that substitutions are made only in the line or lines addressed by *r*.

Some Useful Facts About SUBSTITUTE

It is possible to substitute nothing for any desired characters, thus deleting the specified characters throughout the text area. This is done by typing a Control D immediately after EDITOR prints the first double quote in the SUBSTITUTE command.

Example

```
* /
LINE ONE
LINE TWO
LINE THREE
* SUBSTITUTE ↵
"DC" FOR "LINE DC"
WAIT? N
3
* /
ONE
TWO
THREE
*
```

The SUBSTITUTE command may also be used to determine the number of occurrences of any character or group of characters in the text, without changing the text. This is accomplished by simply substituting the character or group for itself.

Example

```
* SUBSTITUTE ↵
"EDC" FOR "EDC"
WAIT? N
27                       There are 27 E's in the text.
* SUBSTITUTE ↵
"A(DC" FOR "A(DC"
WAIT? N
14                       There are 14 occurrences of
*                        A(.
```

After both forms of the SUBSTITUTE command, the current line, addressed by ., is defined as the last line in which substitutions were made.

The control characters A^C, Q^C, W^C, and I^C may be used during the SUBSTITUTE command before WAIT? is asked. The other editing control characters

discussed under *Editing Text Within A Line* may not be used during this command, however.

There are two control characters, Control G and Control V, which are especially useful with SUBSTITUTE and FIND. In particular, it is possible to substitute Line Feeds or Carriage Returns for other characters, or vice versa, by preceding each Line Feed or Carriage Return with a Control V. Control V and Control G may also be used in other situations, which are discussed below.

CONTROL G

Control G is used to represent any arbitrary character. It is used during

- Line addressing, and
- The SUBSTITUTE command.

When Control G is typed, EDITOR prints an exclamation point.

Example: Control G Used In Line Addressing

```
* [WRITEG^G!G^G!5] DELETE ↵
```

deletes the line containing the word WRITE followed by any two characters followed by a 5.

Control G is especially useful with the FIND command. For example, since the address

:LABELG^G:

refers to any line having a label consisting of the word LABEL followed by any character, the following could occur:

```
* FIND :LABELG^G!:/        This command tells EDI-
LABEL1      A              TOR to print all lines with
LABEL2      B              address :LABELG^G:[1]
LABELA    10
LABEL,
4
*
```

Example: Control G Used With SUBSTITUTE

```
* /
1: LINE ONE
2: LINE TWO
3: LINE THREE
4: LINE FOUR
* SUBSTITUTE ↵
"THIS IS D^G" FOR "G^G!:D^G"
                          THIS IS is substituted for
                          any character followed by
                          a colon.
WAIT? N
4
* /
THIS IS LINE ONE
THIS IS LINE TWO
THIS IS LINE THREE
THIS IS LINE FOUR
*
```

CONTROL V

Control V may be used before a Carriage Return, Line Feed, or control character to cause them to be accepted as text; that is, to inhibit their usual functions. V^G may be used at any time the user desires to inhibit these functions.

Substituting Line Feeds And Carriage Returns

The SUBSTITUTE command may be used to substitute Line Feeds and Carriage Returns for any characters in the text area, and vice versa. Each Line Feed or Carriage Return must be preceded by a Control V to inhibit its usual function (line continuation for the Line Feed, termination for the Carriage Return).

Example

```
* 1/
FIRST:SECOND:THIRD
* SUBSTITUTE ↵
"V^G ↴
D^G" FOR ":D^G"
WAIT? N
2
* 1/
FIRST       Line Feeds have now been substituted for
SECOND      the colons.
THIRD
*
```

If Carriage Returns are substituted for another character, EDITOR will not immediately redefine the lines of text accordingly. However, if the text is written on a file, or punched on paper tape, and read back into EDITOR, the lines will be defined as usual.

Example

```
* 1/
FIRST       The words FIRST and SECOND are fol-
SECOND      lowed by Line Feeds.
THIRD
* SUBSTITUTE ↵
"V^G ↵
D^G" FOR "V^G ↴
D^G"
```

1 - FIND is discussed completely under *The FIND Command*, Page 37.

```
WAIT? N
2
* 1/
FIRST        This is still line 1, even though each phy-
SECOND       sical line ends with a Carriage Return.
THIRD
* WRITE /LINES/↵        The text is written on a file.
NEW FILE↵
7 WORDS.
* 1,$ DELETE↵           Now all text is deleted and
* READ /LINES/↵         the file is read in again.
7 WORDS.
* 1/
FIRST        The lines are now defined as usual.
* 2/
SECOND
* 3/
THIRD
*
```

THE FIND COMMAND

The FIND command allows the user to execute other commands repeatedly for whatever lines in the text area he chooses to specify. This section on the FIND command explains all the variations of FIND available to the EDITOR user, starting with the simplest forms of the command and proceeding to the most complex.

Basic Forms Of FIND

There are two basic forms of the FIND command:

FIND*a*	and a command	*Executes the specified command for all lines in the text area with address* a.
*r*FIND *a*	and a command	*Executes the specified command for all lines in the range* r *with address* a.

More complicated FIND commands may be formed by using a secondary range and/or by using the conditionals AND, OR, NOT, and ...[1].

Example

```
* /
95008 CAMPBELL CALIF
00001 DES MOINES IOWA
30305 ATLANTA GEORGIA
60601 CHICAGO ILLINOIS
80907 COLORADO SPRINGS COLORADO
00002 GRAND RAPIDS MICH
00003 HOUSTON TEXAS
11563 LONG ISLAND NEW YORK
70118 NEW ORLEANS LOUISIANA
00005 PEORIA ILLINOIS
15230 PITTSBURGH PENN
95803 SACRAMENTO CALIF
00006 TORONTO CANADA
94025 MENLO PARK CALIF
* FIND 'CALIF'/        This FIND tells EDITOR
                      to print all lines with ad-
                      dress 'CALIF'.
95008 CAMPBELL CALIF
95803 SACRAMENTO CALIF
94025 MENLO PARK CALIF
3                     Three lines containing
                      CALIF were found.
* 1,'SACR' FIND 'CALIF'DELETE↵
                      This FIND tells EDITOR
                      to delete all lines in the
                      range 1,'SACR' with ad-
                      dress 'CALIF'.[2]
PRINT?N
2
* /
00001 DES MOINES IOWA
30305 ATLANTA GEORGIA
60601 CHICAGO ILLINOIS
80907 COLORADO SPRINGS COLORADO
00002 GRAND RAPIDS MICH
00003 HOUSTON TEXAS
11563 LONG ISLAND NEW YORK
70118 NEW ORLEANS LOUISIANA
00005 PEORIA ILLINOIS
15230 PITTSBURGH PENN
00006 TORONTO CANADA
94025 MENLO PARK CALIF
*
```

The FIND command works as follows: EDITOR starts searching for the specified address ('CALIF' in the above example) at the beginning of the text area, or at the beginning of the range if FIND has been used with the address of a range. Whenever it finds a line containing the specified address, it executes the command used with FIND, for that line. Then it continues searching through the text area until it finds another line with the specified address and executes the command for that line. EDITOR continues thus until it reaches the end of the text area, or of the range if FIND has been used with the address of a range. Thus, after FIND, the current line, addressed by ., is the

1 - See *FIND with Secondary Range: The TO Feature,* Page 40, and *Conditionals Used With FIND,* Page 41.

2 - See Page 39 for a discussion of the PRINT option available with some of the commands used with FIND, including DELETE.

last line in the text area or the last line in the range, depending on whether or not FIND has been used with the address of a range of lines.

After the command used with FIND has been executed for all lines with the specified address, EDITOR prints the number of **lines** found with this address, and terminates the FIND.

The following rules about FIND should be observed:

- The address following FIND **must** be a line label or a text address. Otherwise, EDITOR will respond with a question mark.[1]
- Spaces between terms of the command are ignored.
- Line Feeds may be used between terms of the FIND command to continue the command provided they are preceded by a Control V.

Commands Used With FIND

FIND may be used with any of these commands:

EDIT
MODIFY
DELETE
INSERT
/
=
←

FIND may also be followed by a Carriage Return without a command, to obtain the number of lines with the specified address.

Example

```
* /
LINE ONE
LINE TWO
LINE THREE
* FIND :LINE: ↵
3
*
```

NOTE: The = and ← commands are discussed fully in Section 4, Utility Commands. Both commands are used with the address of a single line. Briefly, the = command prints the EDITOR line number of the line addressed. The ← command prints a line label address of the line addressed, if this line has a line label beginning in print position 1. Thus, for example, if line 44 in the text area is

JONES 1843 S NORTH STREET

these commands could be used as follows:

```
* 'NORTH'=44
* 44←
JONES
*
```

When FIND is used with the EDIT command, EDITOR prints the specified lines one at a time. After printing a line, it waits for the user to edit it. At this point, the user may proceed just as if he were using the EDIT command; all control characters available during EDIT may be used. After he terminates the edit, EDITOR prints the next line to be edited.

Example

```
* /
THIS IS LINE TWO
THIS IS LINE TWO
THIS IS LINE THREE
THIS IS LINE TWO
* FIND 'TWO' EDIT ↵
THIS IS LINE TWO
ZCETHIS IS LINE ONE ↵
THIS IS LINE TWO
FC
THIS IS LINE TWO
ZCETHIS IS LINE FOUR ↵
3
* /
THIS IS LINE ONE
THIS IS LINE TWO
THIS IS LINE THREE
THIS IS LINE FOUR
*
```

FIND used with MODIFY works just like FIND with EDIT, except that the lines to be edited are not printed.

When FIND is used with INSERT, EDITOR prints the lines addressed one at a time and waits for the user to type the text to be inserted. Two things should be remembered:

- The text typed must be terminated with a Control D.
- The line printed is the line **before** which the text will be inserted.

Example

```
* /
LINE TWO
LINE FIVE
* FIND :LINE:INSERT ↵
```

1 - This address may be modified using the conditionals discussed on Pages 41-45.

```
LINE TWO
LINE ONE ↵
D^C
LINE FIVE
LINE THREE ↵
LINE FOUR ↵
D^C
2
*/
LINE ONE
LINE TWO
LINE THREE
LINE FOUR
LINE FIVE
*
```

When FIND is used with DELETE, =, and ←, EDITOR asks the question PRINT? after the command is given. This question may be answered with either a Y (for Yes) or an N (for No). No Carriage Return need be typed after Y or N since EDITOR returns the carriage. Here are the results of each response:

Command Used With FIND	Result Of Response To PRINT?	
	Y	N
DELETE	The lines found are printed one at a time; EDITOR asks OK? after printing each line. Now, a Y tells EDITOR to delete the line just printed; an N, not to delete it.	All lines found are deleted.
=	EDITOR prints the line number followed by the line itself, for each line found.	EDITOR prints the line numbers of all lines found. *NOTE: The last number in the list is the number of lines found,* **not** *an EDITOR line number.*
←	EDITOR prints each line found, thus making its line labels plainly visible.	EDITOR prints a line label address for each line found, if the line has a line label beginning in print position 1.[1]

Example: FIND With DELETE

```
*/
95008 CAMPBELL CALIF
00001 DES MOINES IOWA
30305 ATLANTA GEORGIA
60601 CHICAGO ILLINOIS
80907 COLORADO SPRINGS COLORADO
00002 GRAND RAPIDS MICH
00003 HOUSTON TEXAS
11563 LONG ISLAND NEW YORK
70118 NEW ORLEANS LOUISIANA
00005 PEORIA ILLINOIS
15230 PITTSBURGH PENN
95803 SACRAMENTO CALIF
00006 TORONTO CANADA
94025 MENLO PARK CALIF
* FIND '0000'DELETE ↵
PRINT?Y
00001 DES MOINES IOWA
OK?Y
00002 GRAND RAPIDS MICH
OK?Y
00003 HOUSTON TEXAS
OK?Y
00005 PEORIA ILLINOIS
OK?Y
00006 TORONTO CANADA
OK?N
5        Five lines with address '0000' were found;
*/       only four lines were deleted.
95008 CAMPBELL CALIF
30305 ATLANTA GEORGIA
60601 CHICAGO ILLINOIS
80907 COLORADO SPRINGS COLORADO
```

1 - If any of the lines found do not have a line label beginning in print position 1, the ← command works somewhat differently than described here. See Section 4, *Utility Commands.*

```
11563 LONG ISLAND NEW YORK
70118 NEW ORLEANS LOUISIANA
15230 PITTSBURGH PENN
95803 SACRAMENTO CALIF
00006 TORONTO CANADA
94025 MENLO PARK CALIF
*
```

FIND With Secondary Range: The TO Feature

A secondary range, of the form

a_1 TO a_2

may be used with both forms of the FIND command. Thus, both of the following are allowed:

FIND a_1 TO a_2, a **and a command**

Executes the specified command for all lines addressed by a *which are found in each secondary range* a_1 *TO* a_2 *in the text area.*

*r*FIND a_1 TO a_2, a **and a command**

Executes the specified command for all lines with address a *found in each secondary range* a_1 *TO* a_2 *in the range* r.

In the above, a_1, a_2, and a are all addresses of a single line. Further, they must all be text addresses or line labels. The secondary range must be separated from the address a by a comma.

The secondary range a_1 TO a_2 includes all lines from the line addressed by a_1 to the line addressed by a_2, inclusive.

Example

```
*/
  PICKING JUST SIX QUINCES, NEW
  FARMHAND PROVES STRONG BUT LAZY.

START OF SECONDARY RANGE 1
        A QUICK BROWN
        FOX JUMPS OVER
        THE LAZY DOG.
END OF SECONDARY RANGE 1

  PICKING JUST SIX QUINCES, NEW
  FARMHAND PROVES STRONG BUT LAZY.

START OF SECONDARY RANGE 2
        A QUICK BROWN
        FOX JUMPS OVER
        THE LAZY DOG.
END OF SECONDARY RANGE 2
* FIND :START: TO :END:,'LAZY'/
        THE LAZY DOG.
        THE LAZY DOG.
2
*
```

Here all lines containing the text LAZY that are also in one of the two occurrences of the secondary range :START: TO :END: are printed. The two lines outside the secondary ranges which contain LAZY are not printed.

A secondary range is not at all the same as the address of a range of lines which may precede the FIND command. If a secondary range is used with FIND, EDITOR searches **all** such ranges for the specified address. If the address of a range of lines is used before the FIND command, **only one** such range will be searched for the specified address. Thus, in the above example, using the address :START:,:END: before the FIND yields the following:

```
*./
END OF SECONDARY RANGE 2
*:START:,:END:FIND'LAZY'=
PRINT?Y
```

Which range is searched depends on the current line at the time the command is given. The search for the first address in the range begins at the line addressed by .+1.

```
7
        THE LAZY DOG.
1
*./
END OF SECONDARY RANGE 1
*:START:,:END:FIND'LAZY'=
PRINT?Y
16
        THE LAZY DOG.
1
```

As already indicated, FIND may be used with both a range r and a secondary range a_1 TO a_2.

Example

```
*1,<PICKING>FIND:START: TO :END:,'LAZY'=
PRINT?Y
7
        THE LAZY DOG.
1
*
```

Conditionals Used With FIND

The following conditionals may be used with FIND:

NOT
AND
OR
...

Before discussing the rules by which these conditionals work, we give an example using two of them:

```
*/
BIG HOUSE
SMALL HOUSE
BIG CAT
BIG DOG
SMALL FISH
*FIND 'BIG' AND NOT 'HOUSE' DELETE ↵
PRINT?N
2
*/
BIG HOUSE
SMALL HOUSE
SMALL FISH
*
```

The lines BIG CAT and BIG DOG contain the text BIG and not the text HOUSE; hence, they were deleted.

The command FIND 'BIG' AND NOT 'HOUSE' DELETE deletes all lines which contain the text BIG and do not contain the text HOUSE.

'BIG' AND NOT 'HOUSE' is an example of a conditional expression. It is legal to use a conditional expression in place of the final address *a* in both forms of the FIND command. Thus,

FIND conditional expression $\begin{matrix}\textbf{and a}\\\textbf{command}\end{matrix}$

r FIND conditional expression $\begin{matrix}\textbf{and a}\\\textbf{command}\end{matrix}$

are both legal forms of FIND. Conditional expressions may be used in conjunction with a secondary range, so that

FIND a_1 **TO** a_2**, conditional expression** $\begin{matrix}\textbf{and a}\\\textbf{command}\end{matrix}$

r FIND a_1 **TO** a_2**, conditional expression** $\begin{matrix}\textbf{and a}\\\textbf{command}\end{matrix}$

are legal. In addition, the conditional NOT may be used to modify either or both of the addresses a_1 and a_2 in the secondary range.[1] Thus, the most general form of the FIND command is as follows, where everything in brackets is optional:

[r] FIND [**[NOT]** a_1 **TO [NOT]** a_2,] $\begin{matrix}\textbf{conditional}\\\textbf{expression}\\\textbf{or}\\\textbf{address } a\end{matrix}$ $\begin{matrix}\textbf{and a}\\\textbf{command}\end{matrix}$

The addresses used in the conditional expression, such as 'BIG' and 'HOUSE' above, must be line labels or text addresses just as the addresses a_1, a_2, and *a* must be.

Rules For Using Conditionals With FIND

1. Using NOT In A Conditional Expression

The conditional NOT may be used with FIND before any line label or text address. For example,

FIND NOT 'DATA' EDIT ↵

allows edit of all lines in the text area which do not have the text DATA in them.

When EDITOR encounters a command of the form

FIND NOT *a* $\begin{matrix}\textbf{and a}\\\textbf{command}\end{matrix}$

it searches all lines in the text area for the label or text comprising *a*. Each line of text is searched from left to right. If the text or label is not found in the line, the condition is satisfied; that is, EDITOR will execute the specified command for that line.

One particularly useful feature of FIND involves the conditional NOT and the control characters G^C and V^C. These may be used to remove blank lines from the text area, as shown in the following example:

```
*/
LINE 1
LINE 2


LINE 3

*FIND NOT 'GC!VC ↵
'DELETE ↵
PRINT?N
4
*/
LINE 1
LINE 2
LINE 3
*
```

There are four lines consisting of a Carriage Return alone in the text area.

This command deletes all these blank lines.

As already mentioned, NOT may be used to modify either address in a secondary range a_1 TO a_2. If NOT modifies a_1, the first line of the range is the first line EDITOR finds not containing the text or label comprising a_1. If NOT modifies a_2, the last line of the range is the first line following the first line in the

1 - See *Rules For Using Conditionals With FIND, Rule 1,* below, for the meaning of NOT in a secondary range.

range (addressed by either a_1 or NOT a_1) which does not contain the text or label comprising a_2.

Example

```
*/
BIG RED HOUSE
SMALL HOUSE
RED CAT
BIG DOG
SMALL FISH
*FIND NOT 'BIG' TO NOT 'CAT', 'RED'/
RED CAT
1
*
```

This is the secondary range searched.

Do not forget that EDITOR searches **all** occurrences of the secondary range for the specified text.

2. Using AND In A Conditional Expression

AND is allowed between any two line label or text addresses. For example,

FIND 'AXT' AND 'TYPE' INSERT ↵

allows inserting text before all lines containing both AXT and TYPE.

When EDITOR encounters a command of the form

FIND a_1 AND a_2 **and a command**

it first searches the lines of text from left to right for the text or label comprising a_1. If a line contains this string, it is searched again, this time for the text or label comprising a_2. If this string is found, the AND is satisfied; EDITOR will execute the specified command for that line.

The addresses on either side of the AND may be modified by a NOT. Thus, the following conditional expressions are permitted:

a_1 **AND NOT** a_2
NOT a_1 **AND** a_2
NOT a_1 **AND NOT** a_2

When EDITOR encounters one of these expressions in a FIND command, its search of the text is the same as for a_1 AND a_2, except that lines are checked for the absence of the string modified by NOT.

Example

FIND NOT :DATA: AND '20.1'/

prints all lines which do not have the line label DATA, but do contain the text 20.1. This example is equivalent to

FIND '20.1' AND NOT :DATA:/

3. Using OR In A Conditional Expression

The conditional OR is an inclusive OR. It is allowed between any two text or line label addresses.

Example

FIND 'CALIF' OR 'COLORADO'/

prints all lines which contain either CALIF or COLORADO, or both.

When EDITOR encounters a command of the form

FIND a_1 OR a_2 **and a command**

it first searches the lines of text, from left to right, for the text or label comprising a_1. If a line contains this string, the OR is satisfied for that line, so no search for the string comprising a_2 is necessary. EDITOR will execute the specified command for that line. If a line does not contain the string comprising a_1, it is searched again, this time for the string comprising a_2. The OR is also satisfied if this string is found; EDITOR will execute the specified command for this line.

As with AND, the addresses on either side of the OR may be modified by NOT. If this is done, the search is the same, except that lines are checked for the absence of the string modified by NOT.

Example

FIND [12] OR NOT :EMP: EDIT ↵

allows editing of all lines which contain the text 12, or which do not have the line label EMP.

4. Using ... In A Conditional Expression

The conditional ... is used to indicate that one address follows another address. It is permitted between any two text addresses, or between a line label and a text address.

Example

```
*/
BIG TREE
BIG HOUSE
HOUSE OF BIGG
SMALL HOUSE
BIG RED HOUSE
*FIND 'BIG' ... 'HOUSE'/
BIG HOUSE
BIG RED HOUSE
2
*
```

This command prints all lines which contain the text BIG followed by the text HOUSE.

When EDITOR encounters a command of the form

FIND a_1 ... a_2 **and a command**

it scans the lines of text as follows: First, EDITOR searches a line from left to right for the string comprising a_1. If this string is found, the rest of the line is searched for the string comprising a_2. If this string is found in the rest of the line, the ... is satisfied; the specified command is executed for that line.

NOT may be used to modify the address on the right of the ...; for example, consider the following:

```
*/
BIG TREE
BIG HOUSE
HOUSE OF BIGG
SMALL HOUSE
BIG RED HOUSE
*FIND 'BIG' ... NOT 'HOUSE'/
```

This command prints all lines which contain the text BIG, but do not contain the text HOUSE following BIG.

```
BIG TREE
HOUSE OF BIGG
2
*
```

▶ *CAUTION: The condition*

NOT a_1 ... a_2

can never be satisfied. To see why this is so, consider this example

```
*/
A B
C B
B C D
X Y Z
*FIND NOT 'A' ... 'B'/
0
*
```

Zero occurrences of NOT 'A' ... 'B' are found.

In this example, EDITOR first scans each line from left to right for the string A. If A is found, the line is rejected. Once EDITOR has determined that a line does not contain the string A, it has scanned the entire line and is positioned at the end of the line. Now EDITOR considers the ... in the command, which tells it to search the rest of the line for the text B. But since EDITOR is now at the end of the line, none of the line remains to be searched. Thus, B cannot be found in the rest of the line; that is, NOT 'A' ... 'B' can never be satisfied.[1]

5. Combining The Conditionals

As many addresses as desired, separated by AND, OR, or ..., may be used in a conditional expression, as long as the total length of the FIND command does not exceed 256 characters. Each address in such expressions may be modified by a NOT; the addresses must all be text addresses or line labels, as usual. Thus, the following are all legal:

FIND a_1 AND a_2 OR a_3 and a command

FIND a_1 OR a_2 AND NOT a_3 and a command

FIND a_1 ... a_2 AND a_3 ... a_4 OR a_5 and a command

6. Evaluation Of Conditional Expressions

EDITOR evaluates conditional expressions from left to right. This rule is important; it implies, for example, that

'A' OR 'B' AND 'C'

designates all lines containing either

A and C
or
B and C

rather than lines containing

A
or
B and C

because the expression is evaluated in the following order:

'A' OR 'B' AND 'C'
1
2

Addresses and conditionals may be grouped with parentheses to change the usual order of operation. Thus,

FIND 'A' OR ('B' AND 'C')/

will print all lines containing either

A
or
B and C

▶ *NOTE: The conditional ... always modifies only the immediately preceding address. Thus*

'A' OR 'B' ... 'C'

is equivalent to

'A' OR ('B' ... 'C')

and not to ('A' OR 'B') ... 'C'. The latter expression is both meaningless and illegal, (see the restrictions on

1 - To find all lines which do not contain A followed by B, the conditional expression 'A' ... NOT 'B' OR NOT 'A' may be used. See rules 5 and 6.

the use of parentheses listed below.) To find all lines containing either A followed by C or B followed by C use

FIND ('A' ... 'C') OR ('B' ... 'C')

Note also that NOT always modifies only the address immediately following it.

Example

The conditional expression

'A' AND NOT 'B' OR 'C' ... 'D'

is equivalent to

'A' AND (NOT 'B') OR ('C' ... 'D')

It designates all lines containing either

A and not **B**

or

C followed by **D**

There are three restrictions on the use of parentheses:

- NOT may never modify a parenthetical group; thus, FIND 'A' AND NOT ('B' OR 'C') is not allowed. To find all lines containing A and neither B nor C, use

  FIND 'A' AND NOT 'B' AND NOT 'C'

- The conditional ... may never modify a parenthetical group; thus, FIND ('A' OR 'B') ... 'C' is not allowed.
- Only one level of parentheses is allowed; that is, nested parentheses are illegal.

Examples

FIND 'ABC' OR ('STE' AND 'Q') OR 'VO'/

prints lines containing ABC, or both STE and Q, or VO.

FIND ('DATA' ... '2') AND ('C' OR 'B')/

prints lines containing both

DATA followed by 2
and
C

or

DATA followed by 2
and
B

7. Abbreviation Of Conditionals

The conditionals may be abbreviated as follows:

N for NOT
A for AND
O for OR
. for ...
AN for AND
ANN for AND NOT

We conclude this section with the examples beginning on the next page, which illustrate the way EDITOR evaluates various conditional expressions.

Example 1: NOT and ...

```
* /
A
B
C

A B
B A
A C
C A
B C
C B

A B C
A C B
B A C
B C A
C A B
C B A
* FIND 'B' ... NOT 'A'/
B
A B
B C
C B
A B C
A C B
C A B
7
* FIND NOT 'A' ... 'B'/
0
* FIND 'A' ... NOT 'B' OR NOT 'A'/
A
B
C

B A
A C
C A
B C
C B

B A C
B C A
C B A
13
*
```

Recall that FIND commands of this form always find no lines.

This command finds all lines which do not contain A followed by B. Note that the two blank lines in the text are among the 13 lines found.

Example 2: AND and OR

```
* /
A
B
C

A B
B A
A C
C A
B C
C B
* FIND 'A' OR 'B' AND 'C'/
A C
C A
B C
C B
4
* FIND 'A' OR ('B' AND 'C')/
A
A B
B A
A C
C A
B C
C B
7
*
```

'A' OR 'B' and 'C' is equivalent to ('A' OR 'B') AND 'C'.

Example 3: OR and ...

```
* /
A
B
C

A B
B A
A C
C A
B C
C B

A B C
A C B
B A C
B C A
C A B
C B A
* FIND 'A' OR 'B' ... 'C'/
```

'A' OR 'B' ... 'C' is equivalent to 'A' OR ('B' ... 'C').

```
A
A B
B A
A C
C A
B C
A B C
A C B
B A C
B C A
C A B
C B A
12
*FIND ('A' ... 'C') OR ('B' ... 'C')/
A C
B C
A B C
A C B
B A C
B C A
6
*
```

Example 4: AND, OR, NOT, and ...

```
* /
A
B
C
D

A B
B A
A C
C A
B C
C B
A D
D A
B D
D B
C D
D C
* FIND 'A' AND NOT 'B' OR 'C' ... 'D'/
A
A C
C A
A D
D A
C D
6
*
```

SECTION 4
UTILITY COMMANDS

In addition to commands for input, output, and editing of text, EDITOR has the following utility commands:

Command	Function
CLEAR	Erases contents of text area.
LINES *n*	Sets *n* lines per page for the PRINT command.
TABS *number list*	Changes tabs from initial 8, 16, 32, 40, 45, 50, 55, 60, 65, 70, to positions specified in *number list*.
a=	Types EDITOR line number of line addressed by *a*.
a←	Types a line label address for line addressed by *a*.
QUIT	Returns to the EXECUTIVE.
↴	Prints the next line, addressed by .+1.
↑	Prints the previous line, addressed by .+1.

The commands LINES, ↴, and ↑ were discussed fully under *Output Of Text From Text Area*, Page 17.

The QUIT command simply returns the user to the EXECUTIVE.

THE CLEAR COMMAND: CLEARING TEXT AREA

This command erases the entire contents of the text area. It is used as follows:

```
* CLEAR ↵
ALL?
```

After the user types CLEAR followed by a Carriage Return, EDITOR responds with ALL?. At this point, typing a Y (for Yes) confirms the command. Typing an N (for No) aborts it. No Carriage Return need be typed after the Y or N, since EDITOR returns the carriage automatically.

THE TABS COMMAND: SETTING TAB STOPS

Up to ten tab stops may be set with the TABS command.

The general form of the command is

TABS number list ↵

where the **number list** consists of the print positions at which tab stops are to be set, separated by commas.

Example

```
* TABS 5,10,15,20 ↵
* APPEND ↵
12345678901234567890 ↵
I^C  *I^C  *I^C  *I^C  * ↵
D^C
* /
12345678901234567890
     *    *    *    *
```

Recall that I^C spaces up to the next tab stop.

In this example, tab stops were set at print positions 5, 10, 15, and 20.

Once the TABS command has been given, EDITOR assumes tab stops at the specified print positions until another TABS command has been given. Even if CLEAR has been given, the last TABS command remains in effect.

If no TABS command has been given, EDITOR assumes tab stops at print positions 8, 16, 32, 40, 45, 50, 55, 60, 65, and 70. *NOTE: Each tab stop specified with the TABS command replaces one previously set tab stop. This means that when fewer than 10 tab stops are set with TABS, the remaining preset tabs will still be set. For example,*

**TABS 5,10,15,20 ↵*

causes the preset tab stops at print positions 8, 16, 32, and 40 to be replaced by new tab stops at print positions 5, 10, 15, and 20, respectively. Therefore, after execution of this command there are tab stops set at print positions 5, 10, 15, 20, 45, 50, 55, 60, 65, and 70.

This command has no effect on the current line.

THE = COMMAND: DETERMINING A LINE NUMBER

The general form of this command is

a=

where *a* is the address of a single line. It prints the EDITOR line number of the line addressed.

Example

```
*/
LINE ONE
LINE TWO
LINE THREE
LINE FOUR
*[TWO]=2
```

There is no need to type a Carriage Return after this command. EDITOR prints the line number as soon as the user types the =.

If the LINES command has been given previously, the = command prints the page number, and the line number on that page, of the line addressed, as well as the EDITOR line number. Thus, if the command

LINES 10

has been given, and the third line on page two is

NOW IS THE TIME

the following may occur:

```
*:NOW:=PAGE 2 LINE 3 ;13
```

After execution of the = command, the current line, addressed by . , is the line whose line number was printed.

THE ← COMMAND: DETERMINING A LINE LABEL

The general form of this command is

a←

where *a* is the address of a single line. If the line addressed has a line label beginning in print position 1, this command prints its line label. If the line does not have such a line label, it prints the line label of the nearest preceding line having a line label beginning in print position 1, plus a number indicating how far away the line is.

Example

```
*/
ONE FIRST
TWO SECOND
   THREE THIRD      This line does not have a line
*"FIRST"←           label beginning in print posi-
ONE                 tion 1.
*"THIRD"←
TWO+1               The nearest line having a line
*                   label in print position 1 is the
                    line with label TWO; this line
                    is one line before the line ad-
                    dressed.
```

If no line preceding the line addressed has a label beginning in print position 1, EDITOR simply returns control to the user.

There is no need to type a Carriage Return after the ← command. EDITOR returns the carriage and types the line label. After the ← command, the current line, addressed by ., is the line addressed by the ← command.

If the LINES command has been given previously, the ← command prints the page number and line number on that page of the line addressed, as well as the line label. Thus, if the LINES command has been given and the line

TOTAL SALES FOR JULY

is the second line on page two, the following may occur:

```
*'JULY'←
PAGE 2 LINE 2 ;TOTAL
```

SECTION 5
ADVANCED EDITOR FEATURES

In this section we discuss the ten buffers which are available for storing text, commands, and control characters. Using these buffers, editing "programs" may be created to simplify greatly the user's editing work.

In the subsection *Use Of Buffers*, we also discuss some advanced editing techniques using buffers and some of the commands discussed in Section 4.

GENERAL DESCRIPTION OF BUFFERS

EDITOR has ten buffers numbered 0 through 9. Information may be entered into these buffers from either the terminal or the text area.

Buffers 1 through 9 hold up to and including 8 lines of information.

Buffer 0 holds up to 80 lines; however, only 8 lines may be entered from the terminal.

The contents of any buffer may be printed on the terminal.

Buffers may be loaded with any characters that can be typed from the terminal; the contents may then serve as a source of commands, text and/or control characters for use in editing the text in the text area.

OUTPUT FROM A BUFFER TO THE TERMINAL

There is only one command for output from a buffer, which prints the contents of the buffer at the terminal. The general form of this command is

*n*BUFFER↵

where *n* is the number of the buffer whose contents are to be printed. When the command is given, EDITOR prints the contents of the buffer surrounded by quotes.

Example

If buffer 3 contains the text

THIS IS IN BUFFER 3

the following may occur:

```
* 3BUFFER↵
"THIS IS IN BUFFER 3"
*
```

This command has no effect on the current line.

The contents of a buffer may be transferred to the text area by using Control B at the appropriate time. See *Use Of Buffers*, Page 50. Thus, other commands for output from a buffer are unnecessary.

INPUT TO A BUFFER

Input From The Terminal

The command for input to a buffer from the terminal has the form

*n*LOAD↵
Text, commands, and/or control characters to be enteredD^C

The command is terminated with a Control D, which should immediately follow the information to be stored. No Carriage Return need be typed preceding the terminating D^C. In fact, if a Carriage Return is typed, it will be stored in the buffer along with the rest of the information typed at the terminal.

Examples

```
* 5 LOAD↵
THIS TEXT IS STORED IN BUFFER 5D^C
* 5 BUFFER↵
"THIS TEXT IS STORED IN BUFFER 5"
* 1 LOAD↵
'READ' EDIT↵                 Here, a command is stored in
D^C                          buffer 1.
* 1BUFFER↵
"'READ' EDIT
"                            The " prints below the com-
*                            mand because the Carriage
                             Return following the com-
                             mand is stored in the buffer.
```

If there is anything already in a buffer when a LOAD is given, the new LOAD erases this and inserts whatever was specified in its place.

The control characters A^C, Q^C, W^C, and I^C are available for editing during LOAD.

This command has no effect on the current line.

NOTE: If too many lines are typed during LOAD, the error message

MAXIMUM LOAD IS: 80 LINES IN 0 AND 8 LINES IN 1 THRU 9

?

will be printed. **Nothing** *will be stored in the buffer.*

Input From The Text Area

There are two commands for entering lines of text in the text area into a buffer.

1. The command

   r;*n*LOAD ↵

   puts the line or lines addressed by *r* into buffer *n*. The lines put into the buffer are not deleted from the text area.

Example

```
*
1
2
3
4
5
6
7
8
9
10
* 1,9;0LOAD ↵
* 0BUFFER ↵
"1
2
3
4
5
6
7
8
9
"
*
```

Note that more than 8 lines may be put into buffer 0 if they are entered from the text area.

This form of LOAD is also destructive in that anything in the buffer before the command is given is erased when the LOAD is given.

This form of LOAD also has no effect on the current line.

2. The command

   r;*n*GET ↵

   puts the line or lines addressed by *r* into buffer *n* and also deletes them from the text area.

Example

```
* /
1
2
3
4
5
6
7
8
9
10
* 5,10;9GET ↵
* 9BUFFER ↵
"5
6
7
8
9
10
"
* /
1
2
3
4
*
```

Note that lines 5 through 10 are no longer in the text area.

After the GET command, the current line, addressed by ., is the line preceding the first line put into the buffer. If the first line in the text area is put into the buffer, the current line is simply the first line of text remaining in the text area. Not surprisingly, this is the same rule by which the current line is defined after DELETE.

Like LOAD, GET is destructive to anything previously stored in the buffer.

ERASING THE CONTENTS OF A BUFFER

The command

*n*KILL ↵

erases the contents of buffer *n*. It is not used very often, since LOAD is destructive. It has no effect on the current line.

Note that the CLEAR command erases the contents of the buffers as well as of the text area.

USE OF BUFFERS

Buffers are used as a source of text, commands, and/or control characters. They are particularly useful when a string of text is required in several places, or when a given command or sequence of commands is

to be executed several times in succession. The information needed repetitively is stored in buffers, and when it is needed, the user may access them with Control B.

Control B

The general form

$B^C n$

is used to take commands, text, and/or control characters from buffer *n*. When Control B is typed, EDITOR prints a cross hatch (#) to indicate its use.

Control B may be used to call a buffer at any time. As soon as the buffer number *n* is typed, the contents of the buffer are called; there is no need to type a Carriage Return after the buffer number.

Often, text, commands, and control characters are all stored in one buffer. The important rule to remember in understanding how buffers work is the following:

- When the contents of a buffer are called with Control B, EDITOR interprets them exactly as if they were typed at the terminal.

Using Control B does not erase the contents of the buffer called.

Text In A Buffer

If EDITOR happens to be expecting text when Control B is used, as, for instance, during APPEND, the contents of the buffer are interpreted as text.

Example

```
* 5LOAD ↵
THIS IS LINE D^C
  APPEND ↵
B^C#51 ↵
B^C#52 ↵
B^C#53 ↵
D^C
* /
THIS IS LINE 1
THIS IS LINE 2
THIS IS LINE 3
*
```

Commands In A Buffer

On the other hand, if EDITOR is awaiting a command, as indicated by the asterisk, EDITOR takes a command from buffer *n* when $B^C n$ is typed. *NOTE: SUBSTITUTE and FIND may not be stored in a buffer.*

Example

```
* /
33.4, 55.6
1, 0
0, 1
44.6, 65.2
33.1, 89.5
1, 0
0, 1
34.2, 33.6
* 3LOAD ↵
:1,:,:0,:DELETE ↵
B^C&B3D^C
* B^C#3
?
* /
33.4, 55.6
44.6, 65.2
33.1, 89.5
34.2, 33.6
*
```

In this example, the user wishes to delete the range of lines

```
1, 0
0, 1
```

everywhere it occurs in the text area. To do this, he loads buffer 3 with the command

```
:1,:,:0,:DELETE ↵
```

By itself, this command would delete only one such range (which range would depend on the location of the current line when the command was given). To execute this command repetitively, the user stores a $B^C 3$ in the buffer. EDITOR prints &B instead of # when B^C is typed during LOAD, to indicate that the control character has been stored in the buffer instead of performing its usual function.[1] The user terminates the LOAD with a Control D. Now, when he calls the buffer using $B^C 3$, EDITOR is awaiting a command, so it takes the first command from the buffer and executes it just as if it were typed at the terminal. One of the specified ranges of lines is deleted. Then EDITOR looks for the next thing in the buffer, which is a $B^C 3$. This tells EDITOR to call the contents of buffer 3 again. Another specified range of lines is deleted and buffer 3 is called. This time, there are no more ranges addressed by

```
:1,:,:0,:
```

1 - See the following discussion on storing control characters in a buffer.

in the text area, so EDITOR prints a ? and returns control to the user.

This example points out two useful features of buffers:

- Control characters may be stored in a buffer.
- A buffer may call itself or another buffer, since Control B may be stored in it.

Control Characters In A Buffer

Most control characters are automatically stored in a buffer instead of performing their usual functions when they are typed during the LOAD command. When one of these control characters is typed, EDITOR prints an ampersand followed by the character typed. For example, F^C prints as &F during LOAD.

The following control characters are **not** automatically stored in a buffer when typed during LOAD. These six control characters may be stored in a buffer by preceding them with a Control V when they are typed, to inhibit their usual functions:

1. The four control characters available for editing purposes during LOAD (A^C, Q^C, W^C and I^C) are not stored in a buffer when typed during LOAD, but instead perform their editing functions. For example, in the following,

   *** 1LOAD** ↵
   .+3NA^C←MODIFY ↵

   the Control A deletes the preceding character; it is not stored in buffer 1.

2. Control D, since it terminates the command, is not stored in the buffer when it is typed during LOAD.

3. Control V must be preceded by Control V to store it in a buffer.

Example

```
* /
BROWNE36
JONES75
SMITH15
THOMAS
* 5LOAD ↵
H^C&HV^CA^C&AV^CA^C&A ↵
B^C&B5D^C
* 1,3MODIFY ↵
B^C#5*/
BROWNE
JONES
SMITH
THOMAS
*
```

In this example, the user wishes to delete the numbers from the end of lines 1 through 3 in the text area. He may use the same sequence of control characters in editing each line, so he puts these characters in a buffer. Control H will copy the old line to the new line, then the two Control A's will delete the last two characters in the new line. The Carriage Return will terminate the edit, and then B^C5 will call buffer 5 again, making the control characters available for the next edit. Note that each Control A is preceded by a Control V to inhibit its usual function.

For the actual edit of lines 2 through 3, the user types

```
1,3MODIFY ↵
B^C#5
```

Control characters are now taken from buffer 5 until all lines have been edited. At this point, EDITOR returns control to the user with an asterisk.

We conclude this section with a complete editing example, which demonstrates a technique for performing the same editing on every other line of text in the text area by storing the command

.+2MODIFY ↵

in a buffer. In general, this technique may be modified to edit every *n* lines in the text area by using .+*n*MODIFY. It may be used to edit every line of text in the text area; however, this is more easily accomplished by using

1,$MODIFY ↵

and storing the control characters and/or text in a buffer, as in the preceding example, or by storing

MODIFY ↵

in a buffer.

Other examples of editing techniques using buffers are found in Section 6 of this manual.

Example

```
- EDITOR ↵
* READ /PAY1/ ↵
53 WORDS.
* /
ADAMS, JOHN,34821
    1.50,35.50,53.25
BENTLEY, DICK,40012
    2.75,40.00,110.00
BROWN, JANE,36115
    3.00,40.00,120.00
UNDERWOOD, SAM,49230
    3.00,40.00,120.00
* 1INSERT ↵
```

```
↵
↵
D^C
* 1LOAD ↵
.+2MODIFY ↵
X^C&X,X^C&X,F^C&FB^C&B1D^C
* 1 ↵
* B^C#1
?
* /


34821
    1.50,35.50,53.25
40012
    2.75,40.00,110.00
36115
    3.00,40.00,120.00
49230
    3.00,40.00,120.00
* 1,2DELETE ↵
* WRITE /PAY2/ ↵
NEW FILE ↵
35 WORDS.
* QUIT ↵
-
```

In this example, the user wishes to delete the alphabetic strings from every other line of text in the text area. Perhaps he wishes to use the data as input to a program that will not read strings. He first inserts two blank lines at the beginning of the text area. Then he loads buffer 1 with the following:

```
.+2MODIFY ↵
X^C,X^C,F^CB^C1
```

He then sets the current line to the first line of text by typing the line number 1 followed by a Carriage Return. Now he calls buffer 1. EDITOR takes the command .+2MODIFY from the buffer. Since the current line is the first line of text and the first two lines of text are the inserted Carriage Returns, the line edited by this command is

ADAMS, JOHN,34821

The X^C, deletes characters up to and including the first comma; a second X^C, deletes the rest of the name. Then the F^C copies the rest of the old line, without printing it, and ends the edit. Now the stored B^C1 calls buffer 1 again. At this point, the current line is defined as the line just edited since a MODIFY has just been executed. Thus, when buffer 1 is called again, .+2MODIFY allows edit of the line

BENTLEY, DICK,40012

EDITOR continues similarly, editing every other line in the text area. When the end of the text area is reached, .+2 is no longer defined, so EDITOR prints a question mark and returns control to the user. The user prints his text, deletes the two "dummy" lines from the beginning, and writes the text on a file.

SECTION 6
EDITING EXAMPLES

In this section we present five editing examples in increasing order of complexity.

The examples in the previous sections of this manual were intended to demonstrate the functions of EDITOR's various commands and control characters. The examples in this section, however, are intended to suggest possible applications of the EDITOR language, as well as to demonstrate various editing techniques.

These examples are reproduced from actual terminal printout to help give the reader the feel of on-line use of EDITOR. Remember, though, the best way to get the feel of on-line text editing is to do it.

CREATING A PROGRAM IN EDITOR

EDITOR is frequently used to create a program in another language, such as Tymshare BATCH FORTRAN. This example illustrates preparing a BATCH FORTRAN program. It shows the user typing his program, doing some simple editing, and then writing the program on a file which will later be used as input to the BATCH FORTRAN compiler.

Note the use of the TABS command to set a tab stop at print position 7 and the use of control characters during APPEND and INSERT as well as during EDIT.

```
-EDITOR
*TABS 7
*APPEND                                  The user types his text. I^C is used to
      OPEN (5,INPUT,/DATA/)              space to the tab stop at print posi-
      ASSIGN 55 TO K                     tion 7.
      CALL EOF(L)
      B=0
2     READ (5,3) A
      B=V←B+A   A^C deletes the preceding character, V.
      GO TO 2
55    WRITE (1,4) B
3     FORMAT(F9.0)
4     FORMAT($ THE SUM IS $,F10.3)
      END
*1INSERT
C     THIS PROGRAM ADDS ANY SEQUENCE OF NUMBERS.
C     INPIT IS FROM A FILE. SUM IS PRINTED AT TERMINAL.
C     INPU                               The user types Y^C to allow an edit of
*'EOF'EDIT                               this line; then he types Z^CPUF^C.
      CALL EOF(L)
      CALL EOF(K    The user types Z^C( and then types
*./                 KF^C.
      CALL EOF(K)
*:4:EDIT
4     FORMAT($ THE SUM IS $,F10.3)
4     FORMAT($ THE SUM IS $,F10.3)←/)   He types H^C N^C/)
*./
4     FORMAT($ THE SUM IS $,F10.3/)
*/
C     THIS PROGRAM ADDS ANY SEQUENCE OF NUMBERS.
```

```
C       INPUT IS FROM A FILE. SUM IS PRINTED AT TERMINAL.
        OPEN (5,INPUT,/DATA/)
        ASSIGN 55 TO K
        CALL EOF(K)
        B=0
2       READ (5,3) A
        B=B+A
        GO TO 2
55      WRITE (1,4) B
3       FORMAT(F9.0)
4       FORMAT($ THE SUM IS $,F10.3/)
        END
*WRITE /SUM/
NEW FILE
89 WORDS.
*QUIT

-
```

RETRIEVING RESUMES

This example illustrates the usefulness of the FIND command for information retrieval. The user has a disk file containing brief resumés of company employees. Using EDITOR, it is very easy to update the file as well as to find an employee with a particular kind of experience.

In the first part of the example, the user appends a new employee's resumé. Note that each physical line is terminated with a Line Feed to insure that the entire resumé is printed when accessed with FIND.

```
-EDITOR
*READ /RESUMES/
292 WORDS.
*APPEND
HENRY C. SMITH DEPT. 32
   B.A. UNIVERSITY OF TEXAS, MATHEMATICS
   SCIENTIFIC PROGRAMMING

   FORTRAN

*'YOUNG' EDIT
MICHAEL YOUNG DEPT. 622
   B.A. ENGLISH LITERATURE, IOWA STATE
   HARDWARE SALES
   BUSINESS PROGRAMMING

   COBOL
   BASIC

MICHAEL YOUNG DEPT. 622
   B.A. ENGLISH LITERATURE. IOWA STATE
<    DIST. MANAGEMENT, MARKETING
>
*./
```

A new employee's resumé is appended. The entire resumé consists of one line made up of six physical lines.

Michael Young's resumé is updated.

The user types Z^C↵ twice, uses E^C to insert some text, then types F^C.

```
MICHAEL YOUNG DEPT. 622
   B.A. ENGLISH LITERATURE, IOWA STATE
   DIST. MANAGEMENT, MARKETING
   HARDWARE SALES
   BUSINESS PROGRAMMING

   COBOL
   BASIC

*WRITE /RESUMES/
OLD FILE
336 WORDS.
*FIND 'SCIENTIFIC' AND 'ALGOL'/
JOHN B. CAREY  DEPT. 3H
   PH D MATHEMATICS YALE UNIVERSITY
   PROJECT MANAGEMENT, APPLICATIONS PROGRAMMING
   OPERATIONS RESEARCH
   SCIENTIFIC PROGRAMMING

   FORTRAN
   ALGOL

DONNA WILKES DEPT. A4
   PH D PHYSICS, STANFORD UNIVERSITY
   PROJECT MANAGEMENT, SYSTEMS PROGRAMMING
   SCIENTIFIC PROGRAMMING
   ON-LINE GRAPHICS
   SOFTWARE DEVELOPMENT

   FORTRAN
   ALGOL
   ASSEMBLY LANGUAGE

2
*FIND 'COMPUTER OPERATIONS' AND NOT 'MANAGE'/
DALE MOSS DEPT. 6
   B.S. ELECTRICAL ENGINEERING, U.C. BERKELEY
   SCIENTIFIC PROGRAMMING
   CIRCUIT DESIGN
   REAL TIME SYSTEMS
   COMPUTER OPERATIONS

   FORTRAN
   BASIC
   ASSEMBLY LANGUAGE

1
*FIND 'PL-1' AND 'BASIC'/
0
*FIND 'PL-1' OR 'BASIC'/
CARL LARSON  DEPT. 4B
   GRADUATE WEST HIGH SCHOOL
   DIVISION MANAGER, SOFTWARE DEVELOPMENT
   COMPILER DESIGN
```

The updated resumés are written on the file /RESUMES/.

The user wants an experienced scientific programmer who knows ALGOL.

He wants someone with a background in operations who is not a manager.

There is no one who knows PL-1 and BASIC, but there are three people who know one or the other.

```
   COMPUTER OPERATIONS
   USED CAR SALES

   FORTRAN
   PL-1
   ASSEMBLY LANGUAGE

DALE MOSS DEPT. 6
   B.S. ELECTRICAL ENGINEERING, U.C. BERKELEY
   SCIENTIFIC PROGRAMMING
   CIRCUIT DESIGN
   REAL TIME SYSTEMS
   COMPUTER OPERATIONS

   FORTRAN
   BASIC
   ASSEMBLY LANGUAGE

MICHAEL YOUNG DEPT. 622
   B.A. ENGLISH LITERATURE, IOWA STATE
   DIST. MANAGEMENT, MARKETING
   HARDWARE SALES
   BUSINESS PROGRAMMING

   COBOL
   BASIC

3
*QUIT

-
```

CREATING A DATA FILE WITH INPUT FROM PAPER TAPE

In this example, the user has a paper tape containing sample data which he wants to use in a Tymshare statistical analysis library program. The tape contains the sample data separated by their frequencies; however, the user wishes to have only the data on the file, without the frequencies. He uses the SUBSTITUTE command to accomplish this. Note how using Control V and Control G with SUBSTITUTE increases the power of this command.

```
-EDITOR
*TAPE
261.4,3,270.8,4,265.4,2,261.4,7,258.1,3
252.1,5,268.3,7,250.3,1,272.3,9,262.8,2,255.5,8
249.6,5,280.9,9,270.3,3,263.2,8,258.3,2,256.3,3
259.3,5,270.1,3,259.3,2,253.2,5,266.4,6
*SUBSTITUTE
",
" FOR "
"
WAIT? N
4
*/
```

Here the user appends a comma to each line of text by substituting a comma followed by a Carriage Return for a Carriage Return. V^C precedes each Carriage Return typed.

```
261.4,3,270.8,4,265.4,2,261.4,7,258.1,3,
252.1,5,268.3,7,250.3,1,272.3,9,262.8,2,255.5,8,
249.6,5,280.9,9,270.3,3,263.2,8,258.3,2,256.3,3,
259.3,5,270.1,3,259.3,2,253.2,5,266.4,6,
*SUBSTITUTE
",," FOR ",!,"
WAIT? N
22
*/
```

The frequencies are now deleted by substituting a comma for a comma followed by any character (G^C) followed by a comma.

```
261.4,270.8,265.4,261.4,258.1,
252.1,268.3,250.3,272.3,262.8,255.5,
249.6,280.9,270.3,263.2,258.3,256.3,
259.3,270.1,259.3,253.2,266.4,
*SUBSTITUTE
"
" FOR ",
"
WAIT? N
4
*/
```

Now the commas are removed from the end of each line.

```
261.4,270.8,265.4,261.4,258.1
252.1,268.3,250.3,272.3,262.8,255.5
249.6,280.9,270.3,263.2,258.3,256.3
259.3,270.1,259.3,253.2,266.4
*WRITE /DATA2/
NEW FILE
45 WORDS.
*Q

-
```

CREATING A DATA FILE FROM SUPER BASIC DATA STATEMENTS

In this example, the user has a SUPER BASIC program in which data is stored in DATA statements within the program. He wishes to change the program to read the data from a disk file. To do this, he first changes the DATA statements to a format acceptable for file input, writes them on a new file, and deletes them from the program. He then adds an OPEN statement to the program (necessary for file input/output) and changes the READ statements (used for DATA statement input) to INPUT FROM statements (used for file input). Last, he stores the edited program on a new file.

Note the abbreviation of the conditionals used with FIND, the use of a buffer with MODIFY, and the selective use of the SUBSTITUTE command.

```
-EDITOR
*READ /PROG/
1468 WORDS.
*FIND 'DATA' ANN 'PRINT' ANN 'REM' ANN '!'=
PRINT?Y
83
711 DATA 5000000,5398278,5792597,6179114,6554217,6914625,7257469
84
712 DATA 7580363,7881446,8159399,8413447,8643339,8849303,9031995
87
713 DATA 9192433,9331928,9452007,9554345,9640697,9712834,9772499
86
714 DATA 9821356,9860966,9892759,9918025,9937903,9953388,9965330
87
715 DATA 9974449,9981342,9986501,9990324,9993129,9995166,9996631
88
716 DATA 9997674,9998409,9998922,9999277,9999519,9999683,9999793
89
717 DATA 9999867,9999915,9999946,9999966,9999979,9999987,9999992
90
735 DATA .5,.75,.9,.95,.99,.999,.9999,.99;99,4E44
8
*5LOAD
&X &X &F&B5
*83,90MODIFY
#5*83,90/
5000000,5398278,5792597,6179114,6554217,6914625,7257469
7580363,7881446,8159399,8413447,8643339,8849303,9031995
9192433,9331928,9452007,9554345,9640697,9712834,9772499
9821356,9860966,9892759,9918025,9937903,9953388,9965330
9974449,9981342,9986501,9990324,9993129,9995166,9996631
9997674,9998409,9998922,9999277,9999519,9999683,9999793
9999867,9999915,9999946,9999966,9999979,9999987,9999992
.5,.75,.9,.95,.99,.999,.9999,.99999,4E44
*83,90WRITE /PROGDATA/
NEW FILE
145 WORDS.
*83,90DELETE
*FIND 'OPEN'/
```

The user wants to know where his DATA statements are. The conditional expression used assures him that no PRINT statements or comments containing the word DATA are found.

When buffer 5 is called during MODIFY, the control characters in buffer 5 delete the SUPER BASIC line number and word DATA from the beginning of each DATA statement.

```
954 OPEN T9, INPUT,2  There is already a file opened as file 2.
1
*1←
10  Line 1 has SUPER BASIC line number 10.
*1 INSERT ──────────────────────  Statement 5, to open /PROGDATA/,
                                 is inserted.
5 OPEN /PROGDATA/,INPUT,1
*SUBSTITUTE  INPUT FROM 1: is substituted for READ.
"INPUT FROM 1:" FOR "READ"
WAIT? Y
360 READ P(L9)
OK?Y
702 READ X(K)
OK?Y
910 PRINT "SAMPLE OBSERVATIONS MAY BE READ FROM ANY FILE OR TYPED"
OK?N
2
*:360:/                          ┐
360 INPUT FROM 1: P(L9)          │
*:702:/                          ├─ The new input statements look fine.
702 INPUT FROM 1: X(K)           ┘
*WRITE /NPROG/
NEW FILE
1314 WORDS.
*QUIT

-
```

CONVERTING FROM FORTRAN IV TO BATCH FORTRAN

In this example, the user has a FORTRAN IV program on a disk file /FOR4/. He reads the program into EDITOR, changes it to a BATCH FORTRAN program, and writes the edited program on the file /BFOR/. Three main steps are involved in the conversion:

1. Deleting FORTRAN IV line numbers.
2. Arranging the program in the fixed format of BATCH FORTRAN.
3. Substituting BATCH FORTRAN's symbol for exponentiation (**) with FORTRAN IV's symbol (↑).

Buffers used with MODIFY and FIND, and the SUBSTITUTE command are used to accomplish these steps. Note the use of Control G and the conditionals AND and NOT with the FIND command.

```
-EDITOR
*R /FOR4/      Note abbreviation of the READ command.
91 WORDS.
*/
10. 30 READ (0,40) X
15. IF(X-0)60,70,60
20. 40 FORMAT(F10.0)
30. 60 G=1
40. 20 A=(2*G↑3+X)/(3*G↑2)
50. IF (ABS[A-G]-1E-7)10,80,80
60. 80 G=A
70. GO TO 20
80. 10 WRITE(1,50)X,A
90. 50 FORMAT(/,$THE CUBE ROOT OF $,F10.5,$ IS $,F10.7,/)
100. GO TO 30
105. 70 STOP
110. END
*1LOAD                          The control characters loaded into buffer 1 will
&X &E       &E&F&B1             delete line numbers and insert 6 spaces in front
*1,$MODIFY                      of each line when buffer 1 is called during the
#1*/                            command 1,$MODIFY.
       30 READ (0,40) X
       IF(X-0)60,70,60
       40 FORMAT(F10.0)
       60 G=1
       20 A=(2*G↑3+X)/(3*G↑2)
       IF (ABS[A-G]-1E-7)10,80,80
       80 G=A
       GO TO 20
       10 WRITE(1,50)X,A
       50 FORMAT(/,$THE CUBE ROOT OF $,F10.5,$ IS $,F10.7,/)
       GO TO 30
       70 STOP
       END                      The control characters now loaded into buffer 1 will delete three
*1LOAD ---------------------    spaces from the lines accessed by FIND; that is, the lines with
&S&S&S&F&B1                     FORTRAN labels.
*FIND<!!>AND NOT<IF>AND NOT<GO>MODIFY
#18     8 lines were found and modified after the user typed B^C1.

*/
```

```
   30 READ (0,40) X
      IF(X-0)60,70,60
   40 FORMAT(F10.0)
   60 G=1
   20 A=(2*G↑3+X)/(3*G↑2)
      IF (ABS[A-G]-1E-7)10,80,80
   80 G=A
      GO TO 20
   10 WRITE(1,50)X,A
   50 FORMAT(/,$THE CUBE ROOT OF $,F10.5,$ IS $,F10.7,/)
      GO TO 30
   70 STOP
      END
*SUBSTITUTE
"**" FOR "↑"
WAIT? N
2
*/
   30 READ (0,40) X
      IF(X-0)60,70,60
   40 FORMAT(F10.0)
   60 G=1
   20 A=(2*G**3+X)/(3*G**2)
      IF (ABS[A-G]-1E-7)10,80,80
   80 G=A
      GO TO 20
   10 WRITE(1,50)X,A
   50 FORMAT(/,$THE CUBE ROOT OF $,F10.5,$ IS $,F10.7,/)
      GO TO 30
   70 STOP
      END
*WRITE /BFOR/
NEW FILE
82 WORDS.
*Q

-
```

*The SUBSTITUTE command above replaced each ↑ that was in this line with **.*

APPENDIX A
ERROR MESSAGES

Bell

EDITOR rings a bell when a control character is typed which tells it to do something impossible. For example, if the user types Control Z followed by a character that does not appear in the rest of the old line, EDITOR rings a bell.

EDITOR also rings a bell the first time the user attempts to ALT MODE during APPEND, CHANGE, EDIT, MODIFY, INSERT, and LOAD, as a warning to the user. If ALT MODE is typed again, the command is aborted. All work done during the command is lost.

EDITOR takes no further action when the bell rings; the user may correct his error immediately and continue whatever he was doing when the bell rang. *NOTE: When Z^C, O^C, X^C, or P^C is followed by a character not appearing in the rest of the line, the control character, as well as the character, must be retyped.*

There are two cases when the ringing bell does **not** indicate an error:

1. Whenever Control G is typed, the bell rings.
2. When EDITOR is called from the EXECUTIVE, the bell rings.

LINE TOO LONG

When a line longer than 256 characters is entered into the text area, this message is printed upon termination of the command used to enter the line. The line entered is cut off so that only part of it is entered into the text area.

MAXIMUM LOAD IS: 80 LINES IN 0 AND 8 LINES IN 1 THRU 9

This message is printed when more than eight lines are entered into buffer *n* during *n*LOAD, upon termination of the LOAD. No information is retained in the buffer.

During *r*;*n*LOAD and *r*;*n*GET, 80 lines may be entered into buffer 0. Thus, during these commands the message is printed if *r* consists of more than 80 lines and text is being entered into buffer 0, or if *r* consists of more than eight lines and text is being entered into one of buffers 1 through 9. In all cases, nothing is retained in the buffer.

Question Mark (?)

EDITOR responds with a question mark when the user types a command it cannot understand, such as a misspelled command, a command containing an incorrect line address, or a READ command containing an incorrect file name.

EDITOR will also respond with a question mark if it is commanded to do something impossible, such as to delete the tenth line when there are only eight lines of text, or print text which has been cleared.

After the question mark, EDITOR gives another asterisk.

TOO MANY LINES

The commands MOVE, COPY, and *a*READ operate on a maximum of 512 lines. If the user attempts to apply one of these commands to more than 512 lines, this message is printed.

TOO MUCH TEXT

If more than 60,000 characters are entered into the text area during any input command, EDITOR terminates the command and prints this message. Text entered up to the point where the command was terminated is retained in the text area.

WHAT?

If anything other than a Y or an N is typed in answer to a question asked by EDITOR, it responds with WHAT?. Then the correct response may be typed. EDITOR's questions include:

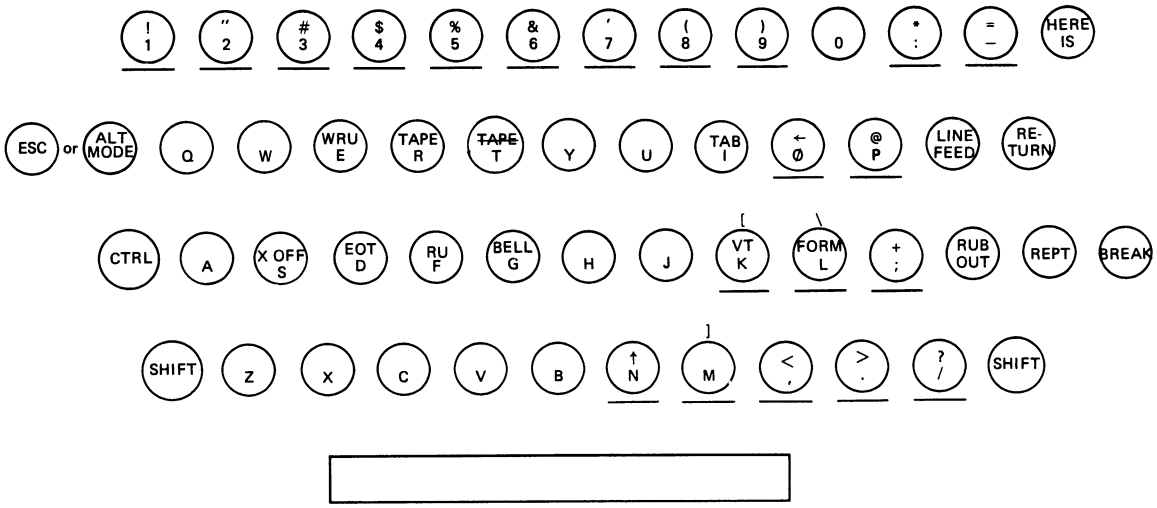
ALL?	during	**CLEAR**
DOUBLE SPACE?	during	**PRINT**
PRINT?	during	**FIND used with = and ←**
PRINT? and **OK?**	during	**FIND used with DELETE**
WAIT? and **OK?**	during	**SUBSTITUTE**

APPENDIX B
DEFINITION OF THE CURRENT LINE

Immediately After All Forms Of:	The Current Line (Addressed By .) Is:
APPEND	Last line appended (=$).
BUFFER	No effect.
CHANGE	Last line typed during CHANGE.
CLEAR	Not applicable.
COPY	First line copied and inserted.
DELETE	Line preceding first line deleted.
EDIT	Last line edited.
FIND	Last line in text area.
*r*FIND	Last line in range *r*.
GET	Line preceding first line of text put into the buffer.
INSERT	Last line typed during INSERT.
KILL	No effect.
LINES	No effect.
LOAD	No effect.
MODIFY	Last line edited.
MOVE	Depends on which way lines are moved.
PRINT	Last line printed.
PUNCH	Last line punched.
QUIT	Not applicable.
READ	Last line in text area.
SUBSTITUTE	Last line in which substitutions were made.
TABS	No effect.
TAPE	Last line read from tape (=$).
WRITE	Last line written on the file.
/	Last line printed. If ALT MODE is typed during command, . does not change.
=	Line addressed by the = command.
←	Line addressed by the ← command.
↵	Line printed by the ↵ command.
↑	Line printed by the ↑ command.

APPENDIX C
THE TERMINAL

THE TERMINAL KEYBOARD[1]



SHIFT
Only those keys which are underlined in the keyboard diagram have a shift position. The SHIFT key operates in the manner of an ordinary typewriter. The SHIFT characters are printed as they appear on the upper half of these keys, with the following exceptions:

SHIFT K = [
SHIFT L = \
SHIFT M =]

CTRL (Control)
Any alphabetic key may be pressed in conjunction with this key. The resulting character, called a control character, does not always print on the terminal. Control characters serve a variety of purposes depending on when they are typed. Some languages, for example, use control characters as editing instructions to the computer. In the Tymshare manuals, a superscript c is used to designate control characters; for example, Control D is shown as D^C. Note the following special control characters:

J^C = Line Feed
M^C = Carriage Return

ALT MODE or ESCAPE
This key is used to abort a command, interrupt the execution of a program, and/or return to the EXECUTIVE. *NOTE: On machines not having either the ALT MODE or the ESCAPE key, use SHIFT K^C.*

HERE IS
Not used in the Tymshare system.

LINE FEED
Advances the paper one line each time it is pressed. When the user is connected to the computer, the system automatically supplies a Carriage Return after every Line Feed.

RETURN (Carriage Return)
Returns the print head to the beginning of a line. The print head goes to the beginning of the **next** line only when the user is connected to the computer; that is, the system automatically supplies a Line Feed after every Carriage Return.

RUB OUT
Used in conjunction with the B.SP. button on the paper tape punch to delete characters punched in error.

REPT (Repeat)
Repeats any character on the keyboard (including a space) when pressed in conjunction with the desired character.

BREAK
DO NOT press this key; it causes a transmission interrupt and possible loss of program and data.

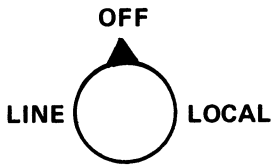
NOTE: Maximum line width is 72 characters on most terminals.

1 - This is the standard terminal keyboard. On individual machines, some keys may not exist or may be located differently than shown in this diagram.

The ON/OFF Controls

The standard ON/OFF control is a three-position dial located on the front of the terminal and to the right of the keyboard.

Standard ON/OFF Control



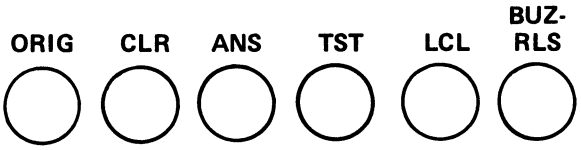
LINE
The terminal is ON and ready to be connected to the computer via the phone line. When the connection is made, the terminal is said to be "on line".

OFF
The terminal is OFF.

LOCAL
The terminal is ready for local ("off line") operations; that is, operations to be performed when the terminal is not connected to the computer. *Paper tape may be punched off line.*

Dial Option

Instead of the ON/OFF controls just described, the terminal may have a six button Dial Option ON/OFF control located to the right of the keyboard:



Dial Option ON/OFF Control

Four of these buttons are used in the Tymshare system:

ORIG
The terminal is ON (Originated) and ready to be connected to the computer.

CLR
The terminal is OFF (Cleared).

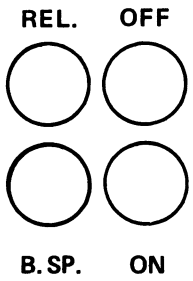
LCL
The terminal is ready for off-line (Local) operations.

BUZ-RLS (Buzzer Release)
Turns off the buzzer which sounds when the paper supply is low.

The Paper Tape Controls

When the terminal is equipped with a paper tape punch and reader, the controls are on the left side of the terminal.

Punch Controls



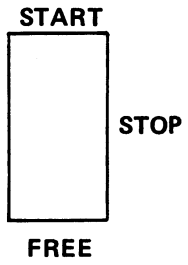
REL.
Releases the paper tape so that the user can pull it through manually.

OFF
Turns the punch OFF.

ON
Turns the punch ON to punch the paper tape.

B.SP.
Back spaces the paper tape one frame each time the button is depressed. Used in conjunction with the RUB OUT key on the keyboard to delete erroneous characters.

Standard Reader Controls



START
Starts and continues paper tape reading.

STOP
Stops paper tape reading.

FREE
Frees the tape reader mechanism so the user can pull the tape through manually.

Optional Reader Control Arrangement

For terminals equipped with the Reader Control Arrangement option, the following control characters have special significance:

Q^C

When typed on the keyboard or sent to the terminal by the computer, Q^C will start the paper tape reader.

S^C

When read from paper tape or sent to the terminal by the computer, S^C will stop the paper tape reader.

NOTE: Before either of these control characters will function, the paper tape reader must be loaded with paper tape and the reader controls must be set to START.

How To Punch Paper Tape Off Line

The user can punch a paper tape while not connected to the computer. Later the tape can be read into the system by means of the READER command of the EXECUTIVE or the TAPE command of EDITOR.

To punch paper tape off line, turn the dial on the front of the terminal to LOCAL and depress the ON button on the paper tape punch controls. Then punch the tape from the keyboard. Note the following special rules:

- Always follow a Line Feed with a Carriage Return.
- Always follow a Carriage Return with a Line Feed.

Example

Off line, type:		On line, type:
THIS IS↵↓	*equivalent to*	**THIS IS**↵
LINE ONE↵↓	*equivalent to*	**LINE ONE**↵

If the user makes a typing error while punching tape, delete the incorrect character immediately by depressing the B.SP. button on the punch controls and then the RUB OUT key on the keyboard. To delete several incorrect characters press B.SP. as many times as necessary and then RUB OUT the same number of times.

APPENDIX D
EDITOR SUMMARY

SUMMARY OF EDITOR LINE ADDRESSES		
Kind Of Address	**Examples**	
	Example	**Function**
EDITOR Line Number	5	Addresses fifth line in text area.
Text Within a Line	'AB, 425' "AB, 425" [AB, 425]	These three addresses address any line containing the text AB, 425.
Line Label	:123:	Addresses line having label 123 which begins in print position 1.
	<END>	Addresses line having label END. This label may be preceded by any number of blanks.
The Last Line of Text	$	Addresses the last line in the text area.
The Current Line	.	Addresses the current line.
Address of a Range of Lines	:34:,"BOB"	Addresses all lines from the line with label 34 (beginning in print position 1) to the line containing BOB, inclusive.
Combination Line Addresses	6:JONES:'SAN JOSE'	Addresses first line containing SAN JOSE following the first line after line 6 that has label JONES.
Address Arithmetic	[DATA]+5	Addresses fifth line after the line containing DATA.
	.-4	Addresses fourth line before the current line.

ALPHABETIC SUMMARY OF EDITOR COMMANDS

a = The address of a single line.

r = The address of a single line or a range of lines.

c = A line label or text address, or a conditional expression.

n = A buffer number.

† = Indicates that the command form may be used without the preceding address *r* to apply to all lines in the text area.

Command	Forms	Function
APPEND	APPEND ↵	Enters text from terminal.
BUFFER	*n*BUFFER ↵	Prints contents of buffer at the terminal.
CHANGE	*r*CHANGE ↵	Changes line or lines addressed to lines typed.
CLEAR	CLEAR ↵	Erases contents of text area and buffers.
COPY	*a*COPY *r* ↵	Inserts a copy of the line or lines addressed by *r* before the line addressed by *a*.
DELETE	*r*DELETE ↵ DELETE ↓	Deletes line or lines addressed. Deletes next line of text.
EDIT	*r*EDIT ↵ EDIT ↓ EDIT ↑	Prints and allows edit of line or lines addressed. Prints and allows edit of next line. Prints and allows edit of previous line.
FIND	*r*†FIND *c* and a command *r*†FIND a_1, TO a_2, *c* and a command	Executes specified command for all lines in specified range satisfying the conditional expression or string address *c*. Executes specified command for all lines satisfying *c* which are also in each secondary range a_1 TO a_2 in the range *r*.
GET	*r*;*n*GET ↵	Enters the line or lines addressed into buffer *n* and deletes them from the text area.
INSERT	*a*INSERT ↵	Inserts lines typed during the command before the line addressed.
KILL	*n*KILL ↵	Erases contents of buffer *n*.
LINES	LINES *m* ↵	Sets *m* lines per page for PRINT.
LOAD	*n*LOAD ↵ *r*;*n*LOAD ↵	Enters information into buffer *n* from the terminal. Enters line or lines addressed into buffer *n*.

Command	Forms	Function
MODIFY	*r*MODIFY ↵	Allows edit of line or lines addressed without printing them.
	MODIFY ↓	Allows edit of next line without printing it.
	MODIFY ↑	Allows edit of previous line without printing it.
MOVE	*a*MOVE *r* ↵	Moves the line or lines addressed by *r* before the line addressed by *a*.
PRINT	*r*↑PRINT ↵	Prints on the terminal in a page format the line or lines specified.
PUNCH	*r*↑PUNCH ↵	Punches line or lines specified on paper tape. EDITOR prints instructions, punches D^C at end of text.
	r↑PUNCH ↓	Punches specified text on paper tape. No instructions are given, no D^C is punched.
QUIT	QUIT ↵	Returns to the EXECUTIVE.
READ	READ ↵	Enters contents of file specified in command into text area.
	*a*READ ↵	Enters contents of specified file into text area before the line addressed by *a*.
SUBSTITUTE	*r*↑SUBSTITUTE ↵ "any charactersD^C" FOR " characters in text areaD^C"	Makes substitutions throughout the line or lines specified.
TABS	TABS number list ↵	Changes tabs from initial settings to print positions specified in number list.
TAPE	TAPE ↵	Enters text from paper tape into text area.
WRITE	*r*↑WRITE ↵	Writes line or lines specified to a file. Multiple blanks are compressed.
	r↑WRITE ↓	Writes line or lines specified to a file. Multiple blanks are not compressed.
/	*r*↑/	Prints line or lines specified at the terminal.
=	*a*=	Prints EDITOR line number of line addressed.
←	*a*←	Prints a line label address for line addressed if the line has a label beginning in print position 1.
Line Feed	↓	Prints next line on the terminal.
↑	↑	Prints previous line on the terminal.

ALPHABETIC SUMMARY OF CONTROL CHARACTERS		
Control Character	**Symbol Printed**	**Function**
A^C	←	Deletes preceding character.
$B^C n$	#*n*	Takes commands, text, and/or control characters from buffer *n* just as if they were typed at the terminal.
C^C		Copies and prints next character in old line.
D^C		During EDIT and MODIFY, it copies and prints the rest of the old line to the new line and ends the edit. During APPEND, INSERT, and CHANGE, it terminates the command if used immediately after a Carriage Return; otherwise it copies the rest of the old line to the new line and ends the new line.
E^C*text*E^C	< >	Inserts text into old line to form new line; first E^C prints <, second E^C prints >.
F^C		Copies but does not print the rest of the old line; ends the edit during EDIT and MODIFY; and ends the line during APPEND, INSERT, and CHANGE.
G^C	!	Used in SUBSTITUTE and line addressing to match any character; for example, XXG^C means XX followed by any character.
H^C		Copies and prints rest of old line and continues the edit at the end of the line.
I^C		Spaces to next tab stop.
Line Feed or J^C		Allows line continuation.
K^C		Deletes next character in old line and prints it.
Carriage Return or M^C		During EDIT and MODIFY, ends the new line and the edit.
N^C	←	Backspaces in the old and new lines.
O^C *and a character*		Copies the old line up to but not including the character typed after it, printing the characters copied.
P^C *and a character*	%	Deletes characters from the old line up to but not including character typed after it.
Q^C	↑	Deletes preceding line.
R^C		Copies and prints rest of old line plus the new line up to the point where R^C was typed; continues edit at this point.
S^C	%	Deletes next character in old line.
T^C		Same as R^C except that it aligns the rest of the old line with the new line.
U^C		Copies and prints the old line up to next tab stop in new line.
V^C		Used before Carriage Return, Line Feed, or a control character to accept them as text (inhibit their usual functions).
W^C	\	Deletes preceding word.
X^C *and a character*	%	Deletes characters from the old line up to and including the character typed after it.

Y^C		Copies but does not print rest of old line and continues edit at the beginning of the new line.
Z^C *and a character*		Copies the old line up to and including the character typed after it, printing the characters copied.

INDEX

NOTE: Page numbers which appear in **bold** *face type refer to those pages where the listed item receives the most detailed discussion.*