FLEX Editor

Technical Systems Consultants, Inc.

TSC 6809 Text Editing System

INTRODUCTION

Contained in the following pages is a complete description of the TSC 6809 Text Editing System. This system is a content-oriented text editor which is powerful, simple to use, and easy to learn. The TSC 6809 Text Editing System is available in a Kansas City Standard cassette version and a FLEX™ 9.0 floppy disk version. This manual, in general, applies to both versions. Commands, actions, and patch points specific to only one version are so indicated.

GETTING THE SYSTEM STARTED

It is recommended that the user read the "Mini-Tutorial" and "Adapting to Your System" sections of this manual before attempting to run the editor.

Disk Version:

The general use of the disk version is completely described in the section of this manual titled "Using the Disk Version".

Cassette Version:

The cassette should be loaded using your system's cassette load routine. After it has been loaded and all of the desired adaptations made, start executing the editor at location 0 (zero). The system should respond with:

```
NEW FILE:
    1.00=
```

The system is now ready to accept the text file input from the keyboard. If the editor is left and later it is desired to reenter the editor to work on the previous text file, it is necessary to enter at location 3, otherwise all workspace will be cleared.

™ FLEX is a trademark of Technical Systems Consultants, Inc.

MINI-TUTORIAL

The purpose of this section is to briefly introduce the reader to the use of the TSC 6809 Text Editing System. We will, therefore, illustrate its use with a number of examples. In order to make it more obvious what things are typed by the user and what things are displayed by the editor, we will subscribe to the convention that things underlined are user-typed and things not underlined are displayed by the editor.

When the editor is initially entered, the response is as shown above. At this time we will create our file by simply typing all lines until finished, terminating each line with a "carriage return".

```
NEW FILE:
    1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
    2.00=TSC TEXT EDITING SYSTEM. A NUMBER OF
    3.00=EXAMPLES WILL BE SHOWN TO ALLOW EASY AND
    4.00=QUICK LEARNING OF ITS FEATURES.
    5.00=FOLLOWING ARE SOME NONSENSE LINES:
    6.00=ABCDEFGHIJKL
    7.00=AAAAAAAA
    8.00=TESTING 1234
    9.00=THIS EDITOR IS FUN TO USE!
   10.00=BBBBBBBB
   11.00=
   12.00=THIS IS THE END OF THIS FILE,
   13.00=AT LEAST FOR NOW.
   14.00=#
   13.00=AT LEAST FOR NOW.
#
```

Notice it was necessary to type a pound sign (#) in column one to leave the buffered input mode. At this time, the system printed the last line and returned with the system prompt (a pound sign). The editor is now ready to accept commands.

Any time characters are being typed into the editor the following two characters have special meaning:

1. "control" H - Deletes the last character typed (backspace).
2. "control" X - Deletes entire current line being typed.

These are useful, when detected typing errors occur, for immediate correction.

Each line of text in the edit file is given or has a line number which is used by the editor to uniquely identify the line. Each line number is of the form "m.nn" where "m" is an integer and "n" represents any of the digits 0 through 9. To specify a line number, one has to specify only that portion of the line number to identify it uniquely. For example, 73, 73., 73.0, and 73.00 may be used to refer to line 73.00; 259.6 refers to line 259.60. The largest line number used with the editor is 9999.99. Let's denote a specification of a line of text by

the symbol "<line>". We will be using this symbol throughout this document.

An editor command tells the editor what action is to be performed and usually what line or block of lines are to be affected (if any). For each editing facility supported by the editor, there is a directive which is used in commands to indicate the desired action. For example, the editor can delete lines of text from a file, insert lines of text into the file, print lines contained in the file, and so on. Corresponding to each capability there is a directive; hence, there is a Delete directive, an Insert directive, a Print directive, and so on. If we define the symbol <directive> to mean any editor directive, the basic from of an edit command is

```
<line><directive>
```

For example, the command to display (Print) line 12.00 is

```
#12P
   12.00=THIS IS THE END OF THIS FILE,
#
```

where "12" is the <line> specification and "P" is the <directive> in this command. As can be seen in the example, this causes line number 12 to be printed on the terminal.

Now, let's learn how to use the insert directive. In normal usage of the word "insert" we say something like, "Insert this card after this other card". To use the Insert directive, we specify the line after which we want to insert new lines followed by an I:

```
<line>I
```

After typing the directive followed by a carriage return, the editor will select an appropriate line number and prompt for input by displaying the line number followed by an equal sign. After each line of text is entered and the carriage return is typed, the editor will prompt for the next line. To exit from the "Insert mode" one simply types a pound sign followed by an edit directive in response to a new line prompt.

Some examples of the use of Insert are

```
#8I
    8.10=THIS IS AN INSERTED LINE.
    8.20=SO IS THIS.
    8.30=#11 I
   11.10=ANOTHER INSERTED LINE.
   11.20=#6 P
    6.00=ABCDEFGHIJKL
#
```

It should be noted that the editor may renumber some lines following the

inserted text. This occurs when enough lines are inserted such that the inserted line numbers overlap line numbers in the original text.

Next, let's learn how to use the Delete directive. With this directive we can delete one line or a block of lines with one directive. To delete only one line, we specify the <line> to be deleted followed by a D:

```
<line>D
```

When the carriage return is typed, the line disappears.

To delete more than one line we need to indicate not only the first line to delete but also the last line to be deleted. Let's call the last line the "target" line and denote its specification as "<target>". Although the editor supports fancier ways to specify the <target>, we'll just consider the two simplest: (1) <target> may be the number of lines to be deleted (counting both the first and last line of the block), or (2) <target> may be a pound sign followed immediately by the line number of the last line of the block to be deleted. Some example <target>s are: 3 (deletes three lines), 26 (delete 26 lines), and #26 (delete lines through line 26.00).

The syntax to Delete a block of lines is

```
<line>D <target>
```

where <line> indicates the first line to delete and <target> indicates the scope of the delete.

To illustrate the use of the Delete directive, let's assume we have a file containing 53 lines with integer line numbers (i.e., 1, 2, 3, ...,53). With the directives

```
#15D
#24D #31
#52D 2
BOTTOM OF FILE REACHED
#
```

we now have a file with lines 1 through 14, 16 through 23, and 32 through 51. The first directive deleted line 15. The second directive deleted lines 24 through 31. The third directive deleted two lines starting with line 52. Since it deleted the last line of the file, the editor displayed the message "BOTTOM OF FILE REACHED".

Before we discuss any more directives, we need to expand the definitions of <line> and <target>.

As editing operations are performed, the editor keeps track of the "current line" which usually is the line most recently affected by a successful edit directive. Upon entering the editor, the "current line" is the first line of the file. If, for example, we have just inserted

three lines between lines 12.00 and 13.00, the current line will be 12.30. One should note that after a line or a block of lines have been Deleted, the line immediately following the last one deleted is made the current line (if the last line of the file was deleted, the new last line of the file will be the current line).

In our discussions above, we have implied that one has to explicitly indicate a <line> for each directive by specifying the line number of the line of interest. However, if <line> is not specified in a directive, the "current line" is used. For example, if one enters the directive

```
#D 2
#
```

the editor will delete two lines starting with the current line. In our example, since we were at line 6.00, the "D 2" operation deleted lines 6.00 and 7.00. As you will learn to appreciate, the "current line" default for <line> is extremely handy.

After performing all of the above operations, our file now looks like this:

```
   1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
   2.00=TSC TEXT EDITING SYSTEM. A NUMBER OF
   3.00=EXAMPLES WILL BE SHOWN TO ALLOW EASY AND
   4.00=QUICK LEARNING OF ITS FEATURES.
   5.00=FOLLOWING ARE SOME NONSENSE LINES:
   8.00=TESTING 1234
   8.10=THIS IS AN INSERTED LINE.
   8.20=SO IS THIS.
   9.00=THIS EDITOR IS FUN TO USE!
  10.00=BBBBBBBB
  11.00=
  11.10=ANOTHER INSERTED LINE.
  12.00=THIS IS THE END OF THIS FILE,
  13.00=AT LEAST FOR NOW.
#
```

We have seen that <line> may be specified by a line number or by default to the current line. There are also several other ways to specify <line>, or in other words, to move the pointer to a desired line prior to the execution of an edit directive. One may also specify <line> with a "+n" or "-n" (where n is an integer) meaning the next nth line in the file or the nth previous line in the file, respectively. Two other useful <line> designators are "↑" ("~" on some terminals) and "↓" ("!" on some terminals). The up arrow "↑" is used to designate the top or first line in the file. The down arrow "↓" is used to move to the last line or bottom of file. These various <line> specifiers are shown in the example below with the PRINT directive.

```
#↑P
   1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
```

```
#+3 P
    4.00=QUICK LEARNING OF ITS FEATURES.
#! P
   13.00=AT LEAST FOR NOW.
#-2P
   11.10=ANOTHER INSERTED LINE.
#
```

There may be times while editing a file when we know part of the contents of a line of interest but don't know its line number nor its displacement from the current line. In such a case we can use the "content-oriented" feature of the editor to find it. The syntax to specify <line> in this way is

/<string>/

where "/" is a character to delimit (enclose) the <string> which is a sequence of characters known to be in the line. When <line> is specified as "/<string>/", the editor will search from the current line through the file to find the next line containing the specified <string>. Some examples will help to clarify this: (1) /PRINT/ denotes the next line containing the character string "PRINT", and (2) /GO TO 35/ refers to the next line containing "GO TO 35". If the <string> is found in any subsequent line of the file, that line will be made the current line and the requested edit operation will be performed on it. If the <string> does not occur anywhere subsequent in the file, the editor will issue the message "NO SUCH LINE" and will not change the current line pointer. Note that the delimiter does not need to be a slash; it may be some other character such as a quote (') or a comma. For example, 'A/B' refers to the next line containing "A/B".

It is also possible to prefix the string designator with "-" (minus sign) to indicate a previous line containing that string. A few examples with our TEST FILE will show the use of "/<string>/" as a <line> designator.

```
#-/QUICK/P
    4.00=QUICK LEARNING OF ITS FEATURES.
#;123; P
    8.00=TESTING 1234
#+'END' P
   12.00=THIS IS THE END OF THIS FILE,
#
```

To summarize, we have seen that <line> may be specified a number of ways, namely: (1) by default to the current line, (2) by typing a line number, (3) by "+n" denoting the nth subsequent line, (4) by "-n" referring to the nth previous line, (5) by /<string>/ denoting the next line in the file containing the indicated string of characters, (6) "-/<string>/" to denote the nearest previous line containing the specified character string, (7) "↑" ("^" on some terminals) to denote the first line of the file, and (8) "↓" ("!" on some terminals) to denote the last line of the file.

Now lets turn our attention to expanding the definition of <target>. As you may recall, a <target> is used in some directives to indicate the number of lines to be affected by the edit operation. We have already seen that a <target> may be specified by (1) an integer "n" indicating the number of lines to be affected, as P3, meaning print 3 lines, and (2) a line number preceded by a pound sign (#) indicating the line number of the last line to be affected, as P #6, meaning print all lines to and including line #6. The <target> is simply a designator telling how many lines the edit directive should operate on. In addition to the two mentioned forms of <target>, we also have, (3) if no <target> is specified in a command whose syntax includes one, a <target> of 1 is assumed, thereby affecting only one line. As with <line>, one may specify <target> by (4) "/<string>/" which indicates the next line in the file containing the specified character string, (5) "↑" to denote the top line in the file, and (6) "↓" to denote the bottom line in the file. A minus sign may be used to indicate that processing is to proceed backward through the file in the following two cases: (7) "-n" and (8) "-/<string>/".

With an understanding of <line> and <target> we can now discuss some more directives. The Print directive is used to display a line or a group of lines. Its syntax is

<line>P <target>

where "<line>" and "<target>" may be specified in any of the ways discussed above. To print just one line one needs to specify only <line> followed by a carriage return; therefore, the following two directives perform the same thing;:

<line>P

and

<line>

Going back to our test file, we can illustrate the various forms of <target> as used with the Print directive.

```
#2P
    2.00=TSC TEXT EDITING SYSTEM. A NUMBER OF
#-1
    1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
#P /EASY/
    1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
    2.00=TSC TEXT EDITING SYSTEM. A NUMBER OF
    3.00=EXAMPLES WILL BE SHOWN TO ALLOW EASY AND
#! P -3
   13.00=AT LEAST FOR NOW.
   12.00=THIS IS THE END OF THIS FILE,
   11.10=ANOTHER INSERTED LINE.
#-/BBB/ P -/123/
   10.00=BBBBBBBB
```

```
     9.00=THIS EDITOR IS FUN TO USE!
     8.20=SO IS THIS.
     8.10=THIS IS AN INSERTED LINE.
     8.00=TESTING 1234
#12P!
    12.00=THIS IS THE END OF THIS FILE,
    13.00=AT LEAST FOR NOW.
#
```

The first directive displayed line 2.00 and made that line the current line. The second directive requested that the line immediately preceding the current line be displayed. The third directive displayed the block of lines from the current line down through the line containing the character string "EASY". The fourth directive printed 3 lines starting at the bottom of the file and ending at line 11.10, which becomes the current line. The fifth directive requested the previous line containing the character string "BBB" be found, and then starting with that line, display all lines going backwards through the file until a line containing the character string "123" has been displayed. This shows the extreme usefulness and power of the content-oriented characteristic of the editor. The last directive requested that all lines from line 12.00 to the end or bottom of file be displayed.

The next directive to discuss is Next which is used primarily to move the line pointer. Although it may be used otherwise, usually it is used only with the default <line>. Its syntax is

N<target>

This directive finds the line indicated by <target>, displays it, and makes it the current line. A few examples will illustrate its use.

```
#↑P
    1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
#N
    2.00=TSC TEXT EDITING SYSTEM. A NUMBER OF
#N 6
    8.20=SO IS THIS.
#N -2
    8.00=TESTING 1234
#
```

The following directive performs single-line replacements or inserts. Its syntax is

<line>=<text>

where "<line>" specifies the number of the line to be replaced or inserted and may, of course, default to the current line. "<text>" is the text to comprise the line. To illustrate this directive, let's continue our example series.

```
#=REPLACE CURRENT LINE HERE
```

```
#5.25=THIS LINE CREATED WITH "EQUALS".
#
```

The first directive changed the contents of line 8.00, the current line. The second example inserted a line with the line number 5.25.

The next directive to be discussed is the Change directive. It is used to change occurrences of one character string into another. Its syntax is

<line>C/<string>1/<string>2/ <target> <occurrence>

where "/" is a delimiter character to separate the two character strings; "<string>1" is the character string to be replaced; "<string>2" is the string of character to replace them; "<target>" specifies the range of the changes; and "<occurrences>" specifies which occcurrence(s) of <string>1 should be replaced in the line(s). If <occurrence> is 1 or is not specified, then only the first occurrence of <string>1 in any line of the block will be changed - the second or subsequent occurrence of the string in such a line will not be affected. If 2 is specified for <occurrence>, then only the second occurrence of <string>1 in any line of the block will be changed. To change all occurrences of the indicated string in the block, use an asterisk (*) for <occurrence>. Let's illustrate the Change directive by continuing our example.

```
#4C/QUICK/FAST/
    4.00=FAST LEARNING OF ITS FEATURES
#8.1C/THIS IS //
    8.10=AN INSERTED LINE.
#-5C;A;$; ;SOME; *
    3.00=EX$MPLES WILL BE SHOWN TO $LLOW E$SY $ND
    4.00=F$ST LE$RNING OF ITS FE$TURES.
    5.00=FOLLOWING $RE SOME NONSENSE LINES:
#12C/E/?/ -2 3
   12.00=THIS IS THE END OF THIS FIL?,
   11.10=ANOTHER INSERT?D LINE.
#
```

The first example replaced the string "QUICK" with the string "FAST" in line 4.00. The second example deleted the string "THIS IS" and a blank from line 8.1. The third example starts at the fifth previous line (line 3.00) and changes every occurrence of "A" to "$" down through all lines until the line containing the character string "SOME" (line 5.00) is reached. The last example changes the third occurrence of "E" to "?" in line 12.00 and then in line 11.10.

The last directive to be discussed is used to exit from the editor. This can be done several ways: STOP, S, or LOG. This will return you to your system monitor.

Now lets go back to our test file and illustrate some of the features and directives we have discussed.

```
#↑P!
    1.00=THIS IS AN EXAMPLE OF THE FANTASTICALLY USEFUL
    2.00=TSC TEXT EDITING SYSTEM. A NUMBER OF
    3.00=EX$MPLES WILL BE SHOWN TO $LLOW E$SY $ND
    4.00=F$ST LE$RNING OF ITS FE$TURES.
    5.00=FOLLOWING $RE SOME NONSENSE LINES:
    5.25=THIS LINE CREATED WITH "EQUALS".
    8.00=REPLACE CURRENT LINE HERE
    8.10=AN INSERTED LINE.
    8.20=SO IS THIS.
    9.00=THIS EDITOR IS FUN TO USE!
   10.00=BBBBBBBB
   11.00=
   11.10=ANOTHER INSERT?D LINE.
   12.00=THIS IS THE END OF THIS FIL?,
   13.00=AT LEAST FOR NOW.
#2C/C /C 6809 /
    2.00=TSC 6809 TEXT EDITING SYSTEM. A NUMBER OF
#/BBB/
   10.00=BBBBBBBB
#-;THIS IS; C 'E'XX' !
    1.00=THIS IS AN XXXAMPLE OF THE FANTASTICALLY USEFUL
    2.00=TSC 6809 TXXXT EDITING SYSTEM. A NUMBER OF
    3.00=XXX$MPLES WILL BE SHOWN TO $LLOW E$SY $ND
    4.00=F$ST LXX$RNING OF ITS FE$TURES.
    5.00=FOLLOWING $RXX SOME NONSENSE LINES:
    5.25=THIS LINXX CREATED WITH "EQUALS".
    8.00=RXXPLACE CURRENT LINE HERE
    8.10=AN INSXXRTED LINE.
    9.00=THIS XXDITOR IS FUN TO USE!
   11.10=ANOTHXXR INSERT?D LINE.
   12.00=THIS IS THXX END OF THIS FIL?,
   13.00=AT LXXAST FOR NOW.
#N -4
   10.00=BBBBBBBB
#-1 I
    9.10=TEST-TEST-TEST
    9.20=1234567890
    9.30=#D!
BOTTOM OF FILE REACHED
#↑D!
BOTTOM OF FILE REACHED
#↑P!
#S
```

The previous tutorial has been only a brief introduction to the TSC 6809 Text Editing System. The remainder of this manual contains a detailed description of each directive with examples, in the next section, followed by "How to Use Tape" and "Using the Disk Version". It is important to read and study the entire manual in order to fully understand and utilize all of the power and features of this editor.

EDITOR DIRECTIVES

The following sections more explicitly describe all the editor commands, use of special features and adapting to your system. You would be well advised to first read the Mini-Tutorial preceding this section. It will give you an overall feel for what the editor can do, thus making the detailed descriptions more understandable. Before getting into the complete descriptions of the editor directives, a few general points will be covered.

USING STRINGS:

Several of the editor directives use character strings as arguments. These arguments are either matched against strings in the text, or replace a string in the text. A string argument begins after a delimiter character and continues as a sequence of any legal characters until the delimiter is again encountered. The delimiters are not considered part of the string to be used in the matching or replacement operations. Although the delimiters in the following descriptions are frequently represented as slashes, "/", any non-blank, non-alphanumeric character may be used as the delimiter such as: * / ( ) $ = , . [ ] : ' etc. Note that the following characters may not be used to enclose strings unless they are preceded by either a plus (+) or minus (-) sign: "↑" (denotes first line of file), "!" (denotes last line of file), "-" (denotes target is above current line), and the character denoted by LINO (normally a pound sign) which is used to flag line numbers. The delimiter character is redefined in each new request by its appearance before a string. If two strings exist in one directive (as in the CHANGE directive), the same delimiter character must be used for each string.

All of the editor directives use the <line> information preceding the directive to position the pointer prior to any directive action. The <line> parameter may of course be null, meaning leave the pointer at its current position. All of the following are valid <line> designators:

1. Any number	references a specific line number
2. +n	denotes the nth subsequent line
3. -n	denotes the nth previous line
4. /<string>/	refers to the next line in the file containing the indicated string of characters
5. -/<string>/	refers to a previous line containing the indicated string

6. ↑	denotes the first line of the file
7. !	denotes the last line of the file
8. null	stay at the current line

Many of the editor directives require <target> information. This tells the editor to operate on the "current" line and all other lines in the file up to the line referenced by the <target>. In cases where a <target> is required, leaving it null will make the <target> default to one, meaning only the current line will be affected by the directive. All of the following are valid <target> designators:

1. an integer n	indicates that n lines should be affected by the edit operation
2. #n	denotes the line number of the last line to be affected
3. /<string>/	denotes the next line in the file containing the specified character string
4. -/<string>/	references the previous line containing the indicated string
5. ↑	denotes all lines up to the top of the file
6. !	denotes all lines to the bottom or last line of the file
7. +or- n	indicates that n lines should be affected and in which direction from the current line
8. (null)	defaults to 1 and only the current line is affected

As we have seen, the form <target> is used to specify a range of lines to which the directive will apply. The directive will be applied to each line, starting with the line specified by <line> and continuing until the target is reached.

If a string <target> is specified, the directive will apply to successive lines of text until a line containing the string is reached. Processing proceeds downward in the file unless the target is preceded by a "-" (minus sign), indicating that processing is to occur upward (toward the first line) in the file. Targets may also be preceded by a plus sign (indicating downard movement). If a line number target is specified, processing begins at <line> and proceeds toward the target

line number. Some examples of <target>s are:

```
2
+10
-3
/STRING/
+/STRING TARGET/
-/BACKWARD DISPLACEMENT TO A STRING/
+*ANY DELIMITER WILL WORK FOR STRING*
++EVEN PLUS SIGNS CAN WORK+
#23.00
```

SPECIFYING A COLUMN NUMBER:

Any "/<string>/" descriptor may be postfixed with a column number immediately after the second delimiter to indicate that the preceding string must begin in the column specified to be found. If the column specified is not in the range of the ZONE in effect, the request will be ignored. Some examples are:

```
/IDENT/11
/PROGRAM/77
*LABEL*2
$COMMENT$30
```

USING THE DON'T CARE CHARACTER:

A "Don't Care Character" may be set to allow indiscriminate matches of parts of a string. When this character is placed in a string, any character in the file will automatically match. The Don't Care Character will have its special meaning only in a string being used to search the file. In other words, the Don't Care Character will not act as such in a replacement string such as the second string of a CHANGE command. The Don't Care Character may be effectively disabled by setting it to a null. Assuming we have previously set the Don"t Care Character to a "?", here are some examples:

/A???/	This would match any 4 letter string beginning with A
@03/??/78@	This would match all days in the 3rd month of 1978
/???/9	This would match any 3 letter string starting in column 9

The COMMAND REPEAT CHARACTER:

A special "Command Repeat Character" has been set up in the editor to allow you to exactly repeat the last command in the input buffer. If a command causes an error or changes the contents of the input buffer, an ILLEGAL COMMAND will be reported upon subsequent use of the Repeat character until another repeatable command is entered. The repeat character is originally set to a CTRL R or 12 hex. Some examples of commands which may be useful to repeat are:

PRINT 15	To print a screen of lines at a time
NEXT	Allows you to single step thru the file with one key
↑CO!!	To quickly fill the workspace
FIND/SOME STRING/	If the first string found is not the one desired

USING THE EOL CHARACTER:

The editor supports an "EOL" or "End Of Line" character to allow multiple commands in a single line. INSERT and OVERLAY are exceptions in that they cannot be followed by other commands. The EOL character may be interactively changed using the SET command. An example of EOL use (with EOL set to "$") is:

```
↑D2$P10$T
```

This sequence will delete the first 2 lines of the file, then print the next 10 lines, and finally return the pointer to the top of the file.

USING TABS:

The user may interactively specify a tab character and up to 20 tab stops. The tab character may then be inserted into a line where it will be expanded when the end of the line is received. If tab stops or the tab character have not been previously set, but some character has been used throughout the file as a tab, it can still be expanded by setting it to be the tab character, setting up your tab stops and then using the EXPAND command on the file. Note that if the tab character has been set, subsequent uses of the INSERT or REPLACE commands will cause automatic tab expansion. However if a tab character is added to the file by the use of a CHANGE, APPEND or OVERLAY command, that character will remain intact in the file until the EXPAND command is invoked on the line containing that tab character.

EDITOR DIRECTIVES

There are five groups of editor directives: environment directives, system directives, "current line" movers, edit directives, and tape directives. A complete description of all directives in each group is given below. In the following descriptions, quantities enclosed in square brackets ([...]) are optional and may be omitted. A backslash (/) is used to separate options.

ENVIRONMENT DIRECTIVES:

H[EADER] [<count>]

MEANING:
A header line of <count> columns will be displayed. The heading consists of a line showing the column numbers by tens, followed by a line of the form "123456789012..." to indicate the column number. Columns for which tab stops are set will contain a minus character instead of the normal digit. If a column count is given, it becomes the default such that if just "H" is subsequently typed, that number of columns will be printed.

EXAMPLES:

HEADER 72	Display column number headings for 72 columns
H 30	Display column numbers for 30 columns

NU[MBERS] [OFF/ON]

MEANING:
The line number flag is turned off or on. If the flag is off, then line numbers will never be printed. If neither "OFF" nor "ON" is specified, then the flag will be toggled from its current state.

EXAMPLES:

NUMBERS OFF	Turn line number printing off
NU ON	Turn it back on
NU	Toggle from on to off or from off to on

REN[UMBER]

MEANING:
The "renumber" directive will renumber all of the lines in the current edit file. Lines in the renumbered file will start with line number "1.00" and will have an increment of one. The line which was current before the command will still be the current line after the command (although its number will probably have been changed).

EXAMPLES:

RENUMBER	Renumber the lines in the current working file
REN	

SET <name> = '<char>'

MEANING:
SET is used to define certain special characters or symbols. The <name>s which may be set are:

TAB - the tab character
FILL - the tab fill character
DCC - the "don't care" character for string searches
EOL - the end of line character which may be used to separate several commands on a single line
LINO - the line number flag character which is used to indicate that a target is a specific line number

The default values are: DCC, TAB and EOL are "null"
FILL is "space"
LINO is "#"

(In the disk version, TAB and EOL are initialized to the values set in the FLEX™ Operating System.)

EXAMPLES:

SET TAB='/'	Set the tab character to a slash
SET TAB=''	Disable tabbing by setting the tab character to a null
SET FILL=' '	Set tab fill character to a blank
SET EOL='$'	Set the EOL character to $
SET LINO='@'	Set the line number flag to at sign

TAB [<columns>]

MEANING:
Used to set the tab stops. All previous tab stops are cleared. If no columns are specified, then the only action is to clear all tab settings. Any tab characters occurring beyond the last tab stop are left in the text. The maximum number of tab stops allowed is 20. Tab stops MUST be entered in ascending order.

EXAMPLES:

TAB 11, 18, 30	Set tab stops at columns 11, 18, and 30
TAB 7 72	Set tab stops for a FORTRAN program
TAB	Clear all tab stops

V[ERIFY] [ON/OFF]

MEANING:
The verify flag is turned on or off. The verify flag is used by the directives CHANGE and NEXT (and several others) to display their results. If neither "ON" nor "OFF" is specified, then the flag will be toggled from its current state.

EXAMPLES:

VERIFY OFF	Turn verification off
V ON	Turn it back on

X

MEANING:
"X" is the cursor control command. Any time this command is entered, the editor will issue the 6 special character string previously set up. See "Adapting to Your System" for details.

EXAMPLES:

X	Output cursor control string

Z[ONE] [C1,C2]

MEANING:
ZONE is used to restrict all sub-string searches (FIND, CHANGE, <target>s, etc.) to columns "C1" to "C2" inclusive. Any substrings beginning outside those columns will not be detected. If C1 and C2 are not specified, then the zones will be reset to

their defaults (columns 1 and 136).

EXAMPLES:

ZONE 11, 29	Restrict searches to columns 11 through 29
ZONE	Search columns 1 thru 136

SYSTEM DIRECTIVES:

LOG

MEANING:
Exit the editor.

EXAMPLES:

LOG

S[TOP]

MEANING:
Same as LOG.

EXAMPLES:

STOP

S

"CURRENT LINE" MOVERS:

B[OTTOM]

MEANING:
Move to the last line in the file and make it the current line.

EXAMPLES:

BOTTOM	Make the last line of the file the current line
B	

F[IND] <target> [<occurrence>]

MEANING:
Move the current line pointer to the line specified by <target> and make it the current line. If the VERIFY flag is on (see VERIFY), the line will be printed. If <occurrence> is specified (an unsigned integer or an asterisk), the directive will be repeated <occurrence> times. If <occurrence> is an integer, it must not start in the first column following the second delimiter of a string <target>, as it would then appear to be a column specifier for that string. If no column is to be specified, insert a space after the second delimiter and before the <occurrence> as in the second example given below. An asterisk means all occurrences of the <target> will be found until the bottom or top of the file is reached. If the target is not found, the current line pointer will not be moved.

EXAMPLES:

FIND /STRING/	Find the next line containing the string "STRING"
F/THREE LINES/ 3	Find the next three lines containing the string "THREE LINES"
F/ALL 'TIL BOTTOM/*	Find all following occurrences of the indicated string
F-/PROGRAM/7 *	Find all previous lines which have the word "PROGRAM" starting in column seven

N[EXT] [<target> [<occurrence>]]

MEANING:
The line specified by the target is made the current line. If the VERIFY flag is on, the line will be printed. If <occurrence> is specified, it must be an unsigned integer. It indicates which next occurrence of a line containing the target is to be made the current line. If the target is not reached, the current line pointer will be positioned at the bottom of the file (or top of the file for a negative <target>). If no target is specified, the next line will be made the current line.

EXAMPLES:

NEXT 5	Make the fifth following line the current line
N	Make the next line the current line
N-10	Make the 10th previous line current

N/STRING TARGET/	Make the next line containing "STRING TARGET" to be the current line
N/3RD OCCURRENCE/ 3	Make the third line containing the indicated string the current line

T[OP]

MEANING:
The first line of the file becomes the current line.

EXAMPLES:

TOP	Make the first line of the file the current line

EDIT DIRECTIVES:

A[PPEND] /<string>/ [<target>]

MEANING:
Append the specified <string> just beyond the last character of the current line (and to successive lines until the target is reached). If the string is postfixed with a column number, then append the string beginning at the specified column (rather than at the end of the line). Any characters previously in the line following the specified column will be lost.

EXAMPLES:

APPEND /./	Append a period to the end of the current line
A *HELLO* 2	Append the word "HELLO" to the end of the current line and to the end of the next line.
A/SEQUENCE/73 *END*7	Append the word "SEQUENCE" starting in column 73 of the current line and successive lines until a line containing the characters "END" beginning in column seven is found

C[HANGE] /<string1>/<string2>/ [<target> [<occurrence>]]

MEANING:

Replace the string specified by <string1> with the string specified by <string2>. If no <target> is specified, only the current line is affected. The slashes represent any non-blank delimiter character. <occurrence> is used to specify which occurrence of <string1> is to be replaced in each line. It is either an unsigned integer or an asterisk ("*") signifying that all occurrences of the substring <string1> are to be replaced with <string2>. By default, only the first occurrence will be changed. Note that if <occurrence> is specified, and if changes are to occur to the current line only, then the target should be a 1 (one).

EXAMPLES:

CHANGE /THIS/THAT/	Replace the first occurrence of "THIS" in the current line with "THAT"
C/A/B/ 1*	Change all occurrences of "A" in the current line to "B"
C /FIRST/LAST/10	Change the first occurrence of "FIRST" to "LAST" in the current line and also in the nine following lines
C /NEW/OLD/ /A TARGET/	Change the first occurrence of "NEW" to "OLD" in each line down through the line containing the string "A TARGET"
C ,A,, -10 *	Remove all "A"s in the current line and in the nine preceding lines
C*HELLO*	Delete the character string "HELLO" from the current line

CC[HANGE] [<column>]/<string1>/<string2>/ [<target> [<occurrence>]]

MEANING:

CCHANGE stands for Controlled Change. This command is exactly like the normal CHANGE command except that the user can interactively specify whether each line containing <string1> should actually be changed or left as is. This allows the user to step thru the file and selectively change certain strings. When a line containing <string1> is found, it is displayed at the terminal and the user receives a prompt, "CHANGE?" If it is desired that the line be changed, type a "Y" for yes. A character other than "Y" will cause the line not to be changed. If an "S" is typed, the command will terminate. Other

characters will cause a search for the next line to be changed.

EXAMPLES:

CCHANGE/ALPHA/OMEGA/!*	Perform a Controlled CHANGE on all occurrences of "ALPHA" thru the rest of the file
CC;A;Z;-20 3	Perform a Controlled CHANGE on the third occurrence of "A" in the current and previous 19 lines

CO[PY] [<destination-target> [<range-target>]]

MEANING:
The current line and successive lines until the <range-target> is reached are copied so that they follow the line specified by <destination-target>. The default <destination-target> is 1, thereby causing a copy of the current line to be placed after the next line. The default <range-target> is 1, thereby copying only one line. After the directive is executed, the current line pointer will be set to the new position of the last line copied. Some lines may be renumbered after a copy with no renumbering message issued.

EXAMPLES:

CO #18	Put a copy of the current line after line 18
COPY #3 4	Copy four lines beginning with the current line and place them after line 3
CO /CHECK/ +/RANGE/	After the next line which has the string "CHECK", place a copy of each line starting with the current line through the line containing "RANGE"

D[ELETE] [<target>]

MEANING:
The current line (and successive lines until the target is reached) is deleted. After the directive is executed, the current line will be the line following the last line deleted.

EXAMPLES:

DELETE 5	Delete five lines (the current line and the next four lines)

D	Delete the current line
D /STRING/	Delete lines from the current line through the next line that contains the string "STRING"

EXP[AND] [<target>]

MEANING:
The current tab character is expanded within all lines, beginning with the current line, continuing down to and including the line specified by <target>. Since tabs are normally expanded as lines are inserted into the file, this directive is primarily of use when one has forgotten to define a tab character.

EXAMPLES:

EXPAND 100	Expand 100 lines starting with the current line
EXP	Expand the current line

I[NSERT]

MEANING:
The editor will enter the buffered input mode, prompting with line numbers (unless line numbers have been disabled, see the NUMBERS directive) and insert the lines below the current line. Buffered input continues until a line beginning with the breakpoint character (a pound sign, "#") in column one is received. The characters following the breakpoint character are treated as an editor directive. The editor will try to choose an insertion increment sufficient to insert at least 10 lines, or if that is not possible, the smallest increment possible. The current line pointer is positioned at the last line inserted. It should be noted that the editor may renumber text lines following the inserted text if the inserted line numbers overlap line numbers previously in the file.

EXAMPLES:

INSERT	Accept line input after the current line
I	Same

I[NSERT] <text>

MEANING:

The text (sequence of characters) which immediately follows the separator (or blank) after the directive name will be inserted as a separate line below the current line of the file. The line inserted becomes the current line. It should be noted that the editor may renumber text lines following the inserted text if the inserted line number overlaps line numbers previously in the file.

EXAMPLES:

```
I THIS BELOW THE CURRENT LINE OF THE FILE

INSERT EVERYTHING AFTER THE FIRST BLANK
```

MO[VE] [<destination-target> [<range-target>]]

MEANING:

The current line and successive lines until the <range-target> is reached are moved so that they follow the line specified by <destination-target>. The default <destination-target> is 1, thereby moving the current line after the next line in the file. The default <range-target> is 1, thereby moving only one line. After the directive is executed, the current line pointer will be set to the new position of the last line moved. Some lines may be renumbered after a move with no renumbering message issued. Attempting to move a large block of data may result in the message "NOT ENOUGH ROOM". This is because the MOVE command performs a COPY and then deletes the old text. If there is not enough room for the copy, the move is not possible. If this message is received, break the move operation up into several smaller moves.

EXAMPLES:

MOVE 3	Move the current line down three lines
MO #1 /TARGET STRING/	Move the current line and all lines down thru the line containing "TARGET STRING" after line 1
MO -/PROGRAM/ 5	Move five lines (including the current line) up within the file so that they follow a line containing the character string "PROGRAM"

MO #10 -5	Move the current line and the four previous lines below line number 10

O[VERLAY][<delimiter>]

MEANING:
The current line is printed, then a line of input is accepted from the terminal (the overlay line). The overlay line will be positioned directly beneath the line printed out. Each character of the overlay that is different from the <delimiter> character (which defaults to a blank) will replace the corresponding character in the current line. The overlaid line will be printed if verify is "ON".

EXAMPLES:

```
OVERLAY
  25.00=THIP IS THE CORRENT LUNE.
OVERLAY    S         U       I
  25.00=THIS IS THE CURRENT LINE.
```

O[VERLAY]<d><text>

MEANING:
This directive is similar to the previous form of the OVERLAY directive with these differences: (1) The current line is not printed. (2) The remainder of the directive line (after the delimiter character) is taken as the overlay text.

EXAMPLES:

```
OVERLAY---AT-------------------- NUMBER.
  25.00=THAT IS THE CURRENT LINE NUMBER.
```

P[RINT] [<target>]

MEANING:
Beginning with the current line, lines are printed until the line specified by target is reached. By default, only the current line will be printed.

EXAMPLES:

P	Print the current line
PRINT 5	Print 5 lines starting with the current line
P -10	Print the current line and the

	nine previous lines
PRINT *STRING*	Print all lines down thru the next line containing "STRING"
P -/STRING/	Print all lines up through the next previous line containing "STRING"

R[EPLACE] [<target>]

MEANING:
A DELETE from the current line through the <target> line is performed. The editor then enters the buffered input mode, putting the new lines into the area vacated. It is not necessary to enter the same number of lines as were deleted. The line numbers of the lines inserted will probably not be the same as those deleted. The current line pointer will be positioned at the last line inserted. By default, only the current line will be deleted.

EXAMPLES:

R	Replace the current line
REPLACE 10	Replace 10 lines starting with the current line
R /TARGET STRING/	Replace all lines from the current line through the line containing "TARGET STRING"

=<text>

MEANING:
The "=" directive replaces the current line with the text supplied. The replacement text begins with the first character following the equals sign. The current line pointer is not moved.

EXAMPLES:

```
=THIS IS REPLACEMENT TEXT.
```

(null)

MEANING:
The null directive (i.e., just a carriage return) prints the current line.

TAPE DIRECTIVES:

GAP

MEANING:
Issue a string of 40 null characters to the tape unit.

EXAMPLES:

GAP	Puts leader or gap on tape

READ

MEANING:
The next file present on the tape will be loaded. All the lines read will be appended to the end of the current work file and the last line read will become the new current line.

EXAMPLES:

READ	Get the next file from tape

SAVE

MEANING:
Write the entire current file out to the tape unit. The tape is formatted as shown in the "USING TAPE" section of this manual. The file is terminated with an ASCII "control Z" character.

EXAMPLES:

SAVE	Puts the current file on tape

W[RITE] [<target>]

MEANING:
This directive is much like SAVE. The only difference being that SAVE puts the entire file on tape, while WRITE puts all lines from the current line through the target line onto tape. The same format as SAVE is used on the tape.

EXAMPLES:

WRITE	Write the current line to tape
WRITE #20	Write all lines from the current line thru line #20 out to the tape unit

USING TAPE:

The TSC 6809 Text Editing System contains four tape directives. These can be used with most types of tape devices including paper tape and Kansas City Standard cassette systems. When using SAVE or WRITE, the text is sent out to the tape, one character at a time, in the following form:

TEXT...(C.R.)...TEXT...(C.R.)...TEXT...(C.R.)(CTRL Z)

The "CTRL Z" is the end of file marker. Note that there are no line numbers, line feeds, or null characters put on the tape, so the file is not suitable for displaying on a terminal in this form. When a tape is read back into the editor using the READ command, line numbers are automatically put back in. If there is more data on the tape than will fit in the workspace, the tape will continue to play, ignoring the excess characters. When the CTRL Z is reached, an error message will be issued.

The TSC 6809 Text Editing System does not provide any routines for issuing control characters for "tape on", "tape off", "record on", or "record off". These routines are to be provided by the user; see "Adapting to Your System" for details.

USING THE DISK EDITOR

The TSC 6809 Text Editing System for the FLEX™ Operating System is a content-oriented text editor which is powerful, simple to use, and easy to learn. It is a great tool for creating or editing various types of text files such as files for the Assembler, Text Processor, and various language compilers and interpreters. The FLEX™ version is a "disk to disk" type editor, meaning any size file which will fit on a single disk may be edited, regardless of the amount of user's RAM available (at least 12K is recommended).

DESCRIPTION

The general syntax of the EDIT utility is:

EDIT,<file spec 1>[,<file spec 2>]

The default extension is TXT and the default drive is the working drive. If only <file spec 1> is given, and the name specified does not exist on the disk, a new file with that name will be created. Creating new files in this manner will cause the editor to respond:

```
NEW FILE:
   1.00=
```

If the EDIT command line only has <file spec 1> specified and it is a name which does exist on the disk, that file will be loaded into the edit buffer, and the editor will issue a "#" as a prompt, signifying that the editor is ready to accept user commands. When the editing process is completed, the original file will now have the same name as before editing except the extension will now be BAK, which stands for "backup". If a file existed with the same name and an extension of BAK, the editor will ask:

DELETE BACKUP FILE?

Answering this with a Y will delete the old backup file and create a new one. A response of "N" will return control back to FLEX™. Any other response will cause the question to be asked again. The newly edited file will have the same name as the original, including the extension. The final form of EDIT is similar to the above but allows assigning the new file a specific name, different from the original. The original would then keep its original name. It should be noted that when editing an existing file, a new file is always created, and the original remains intact, even though its name may be changed. Several examples will help clarify the above syntax. Suppose you want to create a file called TEST.TXT (no such name currently exists on the disk). The following command line should be typed:

```
EDIT,TEST
```

The editor would respond with "NEW FILE" and be ready to accept lines of text.

Suppose you have created a file named TEST.TXT and you now wish to edit it (make changes to it). Typing:

```
EDIT,TEST
```

would now load the file TEST.TXT into memory and the editor would be ready to accept commands. When the editing process is completed, and control is returned to FLEX™, the original file "TEST.TXT" will now have the name "TEST.BAK", but its contents will be unchanged. The file containing all of the changes made while in the editor will have the name "TEST.TXT".

If it is still necessary to make more changes to the new file, the same calling procedure may be used. Now, since there is a file called "TEST.BAK" already on the disk, the editor will ask if you want the backup file deleted. If deleted, the same procedure as above will again take place, and you will end up with the old file having the name "TEST.BAK" and the new one "TEST.TXT".

The final form of the EDIT utility is used if you desire to edit a file, but give the new file a new name. If the following was the command line:

```
EDIT,TEST,TEST2
```

the file TEST.TXT would be loaded into the editor, and the new file would have the name TEST2.TXT. This form of the command line should also be used if it is necessary to edit a file from one drive, and put the new file on a different drive. As an example:

```
EDIT,0.TEST,1.TEST
```

would edit the file TEST.TXT on drive 0, and put the new file, TEST.TXT on drive 1. The file TEST.TXT must not already exist on drive 1. Once in the editor, all of the edit commands apply to the FLEX™ version of the editor with a few exceptions. These differences are stated below.

EXITING THE EDITOR

The STOP command (or "S") should be used. The "LOG" command may also be used. After typing STOP, LOG, or S, the editor will automatically finish any old to new disk file transfers. If editing a large file, this process may take a while, so do not expect FLEX™ to immediately issue the prompt after exiting the editor.

The ABORT command will also return control back to FLEX™; however, the new disk file being created will be deleted and the original file being edited will be given its original name. The net effect is as though no

edit session had taken place (with the exception of a previously existing BAK file having been deleted).

SAVE, READ, WRITE, AND GAP

These commands are still supported but may now also be used to transfer files (or parts of files) to and from the disk, as well as tape. The GAP command may only be used with tape. Upon entering one of the above commands, the editor will respond with:

TAPE OR DISK (T-D)?

A response if "D" will then cause the editor to prompt for the disk file name. A response of "T" will cause the editor to load from tape. Any other response will cause the question to be asked again. Consult the full editor manual for further details in using this set of commands. NOTE: The SAVE command will write the entire content of the current edit buffer to the file. Segments of the edit file which have been flushed with the NEW and FLUSH commands (see below) will not be written, nor will segments yet to be read into the buffer.

THE "NEW" COMMAND

The NEW command aids the editor in handling text files larger than what will fit in memory at any one time. When editing a file, the editor will only load memory with as much of the file as will fit. The NEW command tells the editor that you are done editing that portion of the file and wish to load more of the text into memory. The NEW command works as follows: upon typing the command "NEW", the editor will write everything from the top of the current work buffer (the first line currently in memory) down to but not including the "current line" out to the new file on the disk. At this time, if there is any unread part of the original file, as much of it which can be placed in the unused remaining space of the work buffer (memory) will be read in off of the disk. Control will then be transfered back to the editor and the old "current line" will now be the first line available to be edited. NEW can be used anytime during the edit session, but keep in mind that once it has been used, all parts of the file which were above the current line pointer will become inaccesible for the remainder of the editing session, since they have already been written out to disk. The editor can only operate on text in memory (the work buffer), therefore, global editor commands such as CHANGE and FIND will only be global with respect to the text in the buffer, and not the entire file, unless of course, the entire file will fit in the buffer. The NEW command may also be used when creating a new file. While typing lines into the editor, it is possible to fill the buffer and a message will be issued stating "NOT ENOUGH ROOM". If this happens, typing NEW will cause the file from the top, down to the line pointer, to be written out to the disk, thus freeing up that much of the buffer space. Since no old file exists, nothing new will be read in from the disk.

The FLUSH command works the same as the NEW command except that after having written the text to the file, no additional text is read from the file being edited. This command may be used to reduce the amount of text in memory so that long disk or tape files may be read in using the READ command, or large MOVEs or COPYs may be performed.

BUFFER SIZE

The amount of buffer space available is directly proportional to the amount of memory installed in the computer. The more memory installed, the larger the edit buffer will be. The editor automatically adapts to the memory size.

ADAPTING TO YOUR SYSTEM:

The TSC 6809 Text Editing System has been assembled to run on a SWTPC 6809 system using the S-BUG monitor ROM. The cassette version uses as few ROM-based routines as possible so as to allow easy conversion to other monitor ROMs. The disk version is designed to run under the TSC FLEX™ Disk Operating System and is not easily convertible to other disk operating systems. In both versions, there are some adaptations which the user may make to customize the editor for his own desires or needs.

Both versions of the editor are written to be position-independent and will run correctly when loaded at any memory address, without the necessity of modifying any instructions. The default load address is location 0 (zero). This is also the "cold start" or main entry point address. The "warm start", or re-entry address, is at location 3. The patch addresses given below are relative to the default load address. If the editor has been loaded at some address other than the default, the patch addresses must be biased appropriately.

If you are not using the SWTPC S-BUG monitor ROM, you must supply an input character routine, an output character routine, and a monitor return address as described below. The "Input Character Routine", "Output Character Routine", and "Return to Monitor Address" are vectored through extended indirect jumps in the cassette version, and extended jumps in the disk version. The patch addresses given below for these routines are those of the jump instruction itself, not the address portion. When supplying your own routines, replace the instruction with either an extended jump or extended indirect jump, as appropriate to your monitor ROM.

1. INPUT CHARACTER ROUTINE - This routine is called by the editor to input a character from your keyboard into the A register and return. The parity bit is removed. No other registers must be altered except for condition codes. The address of the jump vector for this routine is at location 0006 hex.

2. OUTPUT CHARACTER ROUTINE - This routine is called by the editor to output a character from the A register to your display device and then return. Except for flags, no other registers may be affected. The address of the jump vector for this routine is at location 000A.

3. RETURN TO MONITOR ADDRESS - This is the routine to which the CPU will jump upon the execution of a STOP or LOG command. Generally it should be set to the re-entry address of your system monitor. The jump vector for this routine is at location 000E hex.

4. FULL DUPLEX - If your terminal requires software echo of typed characters and your input routine does not provide this, change the JMP instruction at location 0006 to the corresponding JSR instruction. (If it's a 6E, make it AD. If it's 7E, make it BD and be sure that 0009 contains a NOP (12).) This change assumes your output character routine does not destroy the A register.

5. MEMORY END - This pointer is the address of the last byte available in the edit workspace. It defaults to 4FFF hex in the cassette version, and to the FLEX™ "Memory End" value in the disk version. If you desire to change this value, store the value of the last usable address of the desired edit work space in location 0017 hex in the cassette version, and in the FLEX™ "Memory End" location in the disk version.

6. SYSTEM CHARACTERS

   A) PROMPT CHARACTER - The prompt character is stored at location 0019 in the cassette version, and 0014 in the disk version. It is presently set to a "#" or 23 hex. Change if desired.
   B) DELETE CHARACTER - In the cassette version, the line delete character is stored at location 0016 hex. It is presently an 18 hex. Change if desired. In the disk version the delete character is the FLEX™ delete character and should be changed with the TTYSET utility.
   C) BACKSPACE CHARACTER - In the cassette vesion, the backspace character is stored at location 0012 hex. It is presently an 08 hex. Change if desired. In the disk version, the backspace character is the FLEX™ backspace character and should be changed with the TTYSET utility.
   D) BELL CHARACTER - When the input buffer is overflowed (more than 136 characters typed), the editor outputs a "bell" character. This is stored at location 0013 hex in the cassette version, and 0012 hex in the disk version and is presently set to a "CTRL G" or 07 hex.
   E) REPEAT CHARACTER - The command repeat character is stored at 001A in the cassette version and at location 0015 hex in the disk version. It is presenly set to a "CTRL R" or 12 hex.

7. CURSOR CONTROL CHARACTERS - The editor outputs a string of six control characters upon execution of the 'X' command. These can be set to special cursor control or other control characters. They are presently set to nulls (00). Leave the 04 at the end of the string intact! The string is located at locations 01CA-01CF in the cassette version, and at locations 01D4-01D9 in the disk version.

Adapting for Tape I/O

Because of the many different hardware configurations for tape systems, the editor does not have any default tape I/O routines. An array is provided for the user to patch in addresses of routines appropriate to his configuration. An address of zero in any entry in this array indicates that there is no such routine, and, in effect, acts as a no-operation.

1. TURN ON TAPE READ - This routine should issue the necessary control functions to start tape motion and prepare the tape hardware for read operations. After calling this routine, the editor will delay (see "Tape Delay", below). Put the address of your "Turn On Tape Read" routine in locations 001B-001C in the cassette version, or

0016-0017 in the disk version.

2. TURN OFF TAPE READ - This routine should issue the necessary control functions to terminate tape read operations and stop tape motion. After calling this routine, the editor will delay (see "Tape Delay", below). Put the address of your "Turn Off Tape Read" routine in locations 001D-001E in the cassette version, or 0018-0019 in the disk version.

3. TURN ON TAPE RECORD - This routine should issue the necessary control functions to start tape motion and prepare the tape hardware for write operations. After calling this routine, the editor will delay (see "Tape Delay", below). Put the address of your "Turn On Tape Record" routine in locations 001F-0020 in the cassette version, or 001A-001B in the disk version.

4. TURN OFF TAPE RECORD - This routine should issue the necessary control functions to terminate tape write operations and stop tape motion. After calling this routine, the editor will delay (see "Tape Delay", below). Put the address of your "Turn Off Tape Record" routine in locations 0021-0022 in the cassette version, or 001C-001D in the disk version.

5. INPUT CHARACTER FROM TAPE - This routine should get one character from the cassette, returning it in the A register. No other registers should be destroyed. This routine defaults to the system "Input Character Routine". Put the address of your routine, if you need to change it from the default, in locations 0025-0026 in the cassette version, or in locations 0020-0021 in the disk version.

6. OUTPUT CHARACTER TO TAPE - This routine should put the character in the A register on to the cassette. A register. No registers should be destroyed. This routine defaults to the system "Output Character Routine". Put the address of your routine, if you need to change it from the default, in locations 0023-0024 in the cassette version, or in locations 001E-001F in the disk version.

7. TAPE DELAY - The editor is assembled to delay approximately 2 seconds after calling one of the "turn on" or "turn off" routines. The value controlling this delay is stored at location 0015 hex in the cassette version and at location 0013 hex in the disk version. It is currently set to 06. Setting it to 0 is no delay with larger values causing longer delays.

SYSTEM CHARACTERISTICS:

1. The maximum line number is 9999.99. If more than 9,999 lines are entered, the line number counter will turn over (go back to 0). the editor, therefore, should not be used with files of 10,000 lines or longer. (This is not really a limitation since 10,000 null lines (line number followed by a carriage return) uses up 40K of memory!)

2. When specifying a line number which is less than one, it is imperative that a leading zero be placed before the decimal point. This is so that the line number will be classified as a number rather than a delimiter.

3. The input buffer will hold 136 characters. If more than 136 characters are typed, they will be ignored and a "bell" character output to the terminal. To terminate the line, it is necessary to type the backspace character and then a carriage return.

4. Setting the "tab" character and the "fill" character the same will delete the TAB feature. There is no logical reason to do this.

5. The method of inserting a line at the top of a file is to use the command: "0INSERT" or "0I". Since there is no line numbered 0.00, the insert takes place at the top of the file.