
# TEXT EDITING SYSTEM

## COPYRIGHT NOTICE



## WARRANTEE INFORMATION

The SSB EDITOR is provided AS IS without warrantee. Reasonable care has been taken to insure that the software operates as described in this manual. If you find a situation in which the assembler does not operate as described, please contact Smoke Signal Broadcasting. We will attempt to correct any errors brought to our attention, but we make no gaurantee to do so.

## EDITOR VERSION SELECTION

The disk on which the SSB EDITOR is supplied contains 3 versions of the software. Each version is configured for a corresponding DOS68 system base address. Select a version according to the base address of DOS68 under which the editor will be running. The versions and their corresponding system base addresses are as follows:

```
EDIT .$  DOS68 at $6000
EDITA.$  DOS68 at $A000
EDITC.$  DOS68 at $C000
```

TABLE OF CONTENTS

# INTRODUCTION

The SSB Text Editor for DOS68 is a powerful line and content oriented editor which is simple to use, and easy to learn. The editor accepts both upper and lower case ASCII characters for text data as well as its command set, making it useful for generating text files. The flexibility of the editor lends its use in applications ranging from text preparation in business environments to commercial software development. The editor has the capability of operating upon files as large as the capacity of a disk.

Reading through the manual, you will soon discover the power of the editor. Many of the time saving, advanced features may seem a little confusing at first. The best way to learn the editor's features is to read through the editing commands, and study the "Mini-Tutorial. Try running the editor using the examples to learn the basics, then study the advanced commands to grasp the full power and capabilities of this editor.

To aquaint the user with the file management conventions used by the editor, the name of the file to be edited, unless otherwise specified, defaults to a filename bearing a type 1 extension. Similarly, default backup file names as well as editor temporaries are assigned type 5 and 8 extensions, respectively. DOS68 initially defines a type 1 extension as .TXT, a type 5 extension as .BAK, and a type 8 extension as .TMP. These extension types and their literal equivalents will be used interchangeably throughout the remainder of this manual. This manual reflects information that is compatible with versions 5.0, 5.0A, and 5.0C of the editor.

## INVOKING THE EDITOR

The editor is invoked using the EDIT command, which uses the general form of:

```
EDIT,<INPUT FILE SPEC>,[<OUTPUT FILE SPEC>]
```

The name of the file(s) specified as input or output default to names bearing a type 1 extension. The default filename extension can be overridden by simply defining the extension explicitly. For example, if FILE.A is specified as part of an input or output filespec, then FILE.A will be operated upon rather than FILE.TXT, which bears the type 1 extension. Type 5 extensions are reserved for the default backup file generated by the editor, and the type 8 extension is reserved for editor temporaries used by the editor at run-time. Therefore the use of type 5 and 8 extensions within input or output filespecs is restricted. The drive on which the file(s) will be found defaults to the work drive as assigned at editor run-time. Refer to the SET command documentataion regarding work drive and extension type assignments.

## CREATING A FILE

Creating a new file with the editor is accomplished using a variation of the general command form by ommitting the input file spec. For example:

```
EDIT,,NUFILE
```

will initiate the creation of the file NUFILE.TXT on the current work drive. The editor will display:

```
NEW FILE:
    1.00=
```

indicating its readiness to accept text input from the user. Again, if no filename extension is specified, the editor will by default assign a type 1 extension to the filename.

## UPDATING AN EXISTING FILE

The simplest and most convenient command form for invoking the editor to update an existing file is:

```
EDIT,TEST
```

Since a specific drive is not specified, the drive is assumed to be the DOS68 work drive as defined by the user. The editor will load the file TEST.TXT found on the work drive into the text buffer. After the editing process is completed, control is returned to DOS68. During the DOS68 return process, the editor is transparently performing housekeeping on the work drive. In this example, if the file TEST.BAK exists, it is deleted. Next, the input file TEST.TXT containing the original data is renamed TEST.BAK. Finally, a temporary file named TEST.TMP reflecting all the changes made while in the editor is renamed TEST.TXT. The automatic maintainence of the new copy and old copy simplifies disk management for the user. Note that if additional old versions of TEST.TXT are required, simply rename TEST.BAK before each edit pass.

If you desire to edit a file, but want to give the output file a another name, use the command form:

```
EDIT,TEST,TEST2
```

The file TEST.TXT found on the work drive will be loaded into the text buffer, and the new file TEST2.TXT will be created on the same drive. This same general form is used to edit an input file found on one drive, and create an output file on anotherdrive. For example:

```
EDIT,0:TEST2,1:TEST2.NEW
```

would allow an edit of TEST2.TXT on drive 0, with the output file TEST2.NEW existing on drive 1 at the conclusion of the edit session. Note that the .NEW extension will override the default type 1 extension.

If at some point it is necessary to edit a file bearing reserved extensions such as a types 5 or 8, a qualified output file must be specified. Therefore, if the file TEST.BAK must be edited, a command like:

```
EDIT,TEST.BAK,TEST.OLD
```

should be used. If, however, the user does not specify a qualified output filespec, then the editor display the message:

```
"YOUR INPUT FILE EXTENSION WITHOUT AN
 OUTPUT FILE SPECIFIED, IS NOT ALLOWED"
```

The editor will transfer control back to DOS68 as type 5 and 8 extensions are reserved for the editor.

## DEFERRED FILE INPUT

To invoke the editor using its deferred input mode, use the EDIT command without file specs as shown below:

```
EDIT
```

The editor, once loaded and running, will prompt the user for file specs. with:

```
FILENAME:
```

The editor is now ready to accept file specifications. Having the capability of entering file specifications after the editor has been loaded and started can be advantageous to single drive as well as multi-drive disk system users. The user can for example, remove the disk on which the editor resides, and mount a disk which contains or may yet contain the file(s) which will be operated upon by the editor. The user has virtually "gained" an additional drive on his system!

# EDITOR DIRECTIVES

## DIRECTIVE OVERVIEW

There are four groups of editor directives: environment directives, system directives, "current line" movers, and edit directives. A complete description of all directives in each group is covered in this section. In the following descriptions, quantities enclosed in square brackets ([ ...]) are optional, and may be omitted. Vertical lines (|) are used to separate the options.

## STRING ARGUMENTS

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 legal 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 the buffer),"!" ( denotes the last line of the buffer), "-" (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 preceeding 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 | Denoting the nth subsequent line. |
| 3. | -n | Denoting the nth previous line. |
| 4. | /<string>/ | Referring to the next line in the file containing the indicated string of characters. |
| 5. | -/<string>/ | References 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 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 1, 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 a previous line containing the indicated string.

5. ~ denotes all lines up to the top of the file.

6. ! references all lines down to the bottom or last line of the file.

7. +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 downward 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 WILL WORK+
#23.000
```

COLUMN SPECIFICATION

Any "/<string>/" descriptor may be postfixed with a column number immediately after the delimiter which indicates that the preceding string must begin in the column specified to be found.

If the column specified is not in the rrange of the ZONE in effect, the request will be ignored. Some examples are:

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

REPEATING COMMANDS

The last editor directive can be repeated while in the command mode by typing a control "D". This simplifies repetitive operations by eliminating the need to continually re-enter the directive. The INSERT and OVERLAY directives should not be repeated. Some directives which are useful to repeat are

| | |
|---|---|
| P23 | To print a screen full of lines at a time. |
| NEW | To step through the input file with the touch of a key. |
| F/LABEL/ | To step through the text buffer on each occurrence of the string "LABEL" |

MULTIPLE COMMANDS

The editor supports a user definable EOL or End Of Line character to allow multiple commands to exist on a single command line. Using the muliple command capability of the editor, the user can compose and execute unique, complex editing functions which would otherwise require a series of individual commands entered separately. The EOL character can be defined as well as changed by the user with the editor's SET directive. The INSERT and OVERLAY directives can be used as part of multiple commands, but they cannot be followed by another command. With the EOL set to "$", the following example shows the implementation of a multiple command:

```
T$F/END/$C/END/STOP/$-10P20$T
```

This multiple command moves the line pointer to the top of the buffer, finds the first occurrence of the string "END", changes "END" to "STOP", backs the line pointer back 10 lines and prints the next 20 lines, then restores the pointer to the top of the buffer

TABS

The editor supports tabbing with up to 20 tab stops. The TAB character and the tab FILL character are definable by the user with the editor's SET directive. In using tabs, the tab character is imbedded in the line where it will be expanded when the end of the line is recieved. If the tab stops or the tab character have not been defined, yet some tab character has been propagated thoughout the file, the tabs can be expanded by first defining the tab character to be the same as what exists in the file, and then invoking the EXPAND directive. Note that if the tab character has been set, subsequent use of the INSERT or REPLACE directives will cause automatic tab expansion. If the tab character is added to the file using the APPEND, CHANGE or OVERLAY directives, it will remain in the file until an EXPAND directive expands the lines containing the tab character.

ENVIRONMENT DIRECTIVES

H[EADER] <columns>

MEANING:

A line of <columns> headings will be displayed. The heading is of the form "123456789Ø12..." 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 count such that if just "H" is typed afterwards, that number of columns will be printed.

EXAMPLES:

| | |
|---|---|
| HEADER 72 | Display column number headings for 72 columns. |
| H 3Ø | Display column numbers for 3Ø 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.ØØ" and will have an increment of "1.ØØ". 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 the lines in the current working file. |

| | |
|---|---|
| REN | Renumber the lines in the current working file. |

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,

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.

Setting the TAB character and the FILL character the same will defeat the TAB feature, therefore there is no logical reason to do this.

Setting the EOL character will allow the user to use multiple commands in a single command line. INSERT and OVERLAY cannot be followed by other commands. An example of EOL use (with EOL set to "$") is:

```
D2$P10$T
```

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

The default values are: TAB and EOL are "null".

FILL is "space"

LINO is ""

EXAMPLES:

| | |
|---|---|
| SET TAB = ':' | Set the tab character to semicolon. |
| SET TAB = '' | Disable tabbing by setting the tab character to null. |
| SET FILL = ' ' | Set the tab fill character to a blank. |
| SET EOL = '$' | Set the EOL character to $. |

SET LINO = '@'  Set the line number target escape.

TAB [<COLUMNS>]  10,14,16,25

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.

EXAMPLES:

| | |
|---|---|
| TAB 11, 18, 30 | Set tab stops are columns 11, 18 and 30. |
| TAB 7 72 | Set tab stops for a FORTRAN program. |
| TAB | Clear all tab stops. |

V[ERIFY] [OFF|ON]

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 "MODIFYING THE EDITOR" for details on how to define the string

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 through 136. |

## SYSTEM DIRECTIVES

### CLOSE

MEANING:

CLOSE performs the same text buffer and file transfers as LOG without returning to DOS68. The editor will remain running to allow continuous file editing. The editor will prompt the user for file specifications with the message:

FILES:

EXAMPLE:

```
CLOSE

FILES:LETTER.OLD,1:LETTER.NEW
```

### DOS

MEANING:

The DOS command causes the editor to terminate execution and return control to DOS68. The input file is left un-changed, the temporary work file is deleted, and no output file is generated.

EXAMPLE:

```
DOS
```

### LOG

MEANING:

Conclude an edit session and exit the editor by writing the contents of the text buffer to the output file, and transfer the remainder of the input file to the output file. Control is transfered to DOS68. If the input file is large, the message "BUFFER FULL - COMPLETE FILE NOT READ" will appear, indicating that the editor is utilizing the entire text buffer in the process of transferring the contents of the input file to the output file.

EXAMPLE:

```
LOG
```

## NEW

MEANING:

The NEW command transfers the contents of the text buffer from the top down to but not including, the "current" line into the output file. Next, as much unread data that can be placed into the text buffer will be read from the input file. Note that if the input file is large, the message "BUFFER FULL - COMPLETE FILE NOT READ" may appear. This is an indication that there is still unread data in the input file although the text buffer was filled to usable capacity. Sufficient free space remains in the buffer for additional text storage by the user.

The NEW command may be used in event that the "NOT ENOUGH ROOM" message appears while editing a new or existing file. To remedy the situation, issue a NEW command, and the contents of the text buffer from the top line to the current line will be transferred to the output file, with the current line now the top line in the buffer.

NEW may also 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 inaccessable for the remainder of the session. The editor can only operate on text in the text buffer, therefore, global commands such as CHANGE and FIND will be global only with respect to the text currently in the buffer, and not the entire file, unless of course, the entire file will fit in the buffer.

EXAMPLE:

```
NEW
```

## READ

MEANING:

The READ command gives the user the capability of transferring the contents of a specified disk file into to the end of the current text buffer, and the last line read will become the new current line. The number of lines that will be read are limited only by the amount of free space currently in the text buffer. Upon receipt of a READ command, the editor will respond with:

```
FILE NAME:
```

after which the name of the file to be read is entered. Note that if the entire contents of the specified file does not fit within the free space, the "BUFFER FULL - COMPLETE FILE NOT READ" message will appear.

EXAMPLE:

READ

FILE NAME:1:MODULE.TXT

RETRY

MEANING:

In event of a fatal disk error during a NEW, LOG, or STOP operation, RETRY will create a new output file and attempt to continue the disk I/O operation where it left off. For example, if a disk error 7 occurs, (disk full error), then the message:

CORRECT PROBLEM AND TYPE "RETRY"
TO ABORT AND RETURN TO DOS - TYPE "DOS"

will appear. Typing RETRY will cause the output file to be re-created, and the disk I/O operation to be continued. Note that if DOS is typed, the editor will abort leaving the file that was being edited unchanged.

STOP

MEANING:

Same as LOG.

EXAMPLES:

STOP

WRITE [<target>]

MEANING:

Write all lines from the current line through the target line to a specified disk file. Upon receipt of WRITE, the editor will prompt the user for the file specification of the file with:

FILE NAME:

EXAMPLE:

WRITE *TARGET STRING* Write all lines from the current line
to the line containing "TARGET STRING"
FILE NAME: 1:MYFILE.TXT into the file MYFILE.TXT on drive 1.

## CURRENT LINE CONTROL

### 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 (see VERIFY) is on, the line will be printed. If <occurrence> is specified (an unsigned integer or an asterisk), the directive will be repeated <occurrence> times. An asterisk means all occurrence of the <target> will be found until the bottom or the top of the file is reached. If the target is not reached, 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/PUSH/ * | Find all following occurrences of the word PUSH. |
| F-/PROGRAM/7 * | Find all previous lines which contain the characters "PROGRAM" beginning 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 (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-1Ø | Make the tenth previous line current. |
| N/STRING TARGET/ | Make the next line containing "STRING TARGET" to be the current line. |
| N/THIRD 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.

EDITING DIRECTIVES

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

MEANING:

Append the specified <strings> just beyond the last character of the current line (and to successive 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] /<string> /<string> / [<target> [<occurrence>]]

MEANING:

Replace the string specified by <string> with the string specified by <string> . If no <target> is specified, only the current line is affected. The slashes represent any nonblank delimiter character. <occurrence> is used to specify which occurrence of <string> is to be replaced in each line. It is either an unsigned integer or an asterisk, "*" signifying that all occurrence of the substring <string> are to be replaced with <string> . 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 /FIRST/LAST/10 | Change the first occurrence of "FIRST" to "LAST" in the current line and also in the nine follow- |

ing 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,, -1Ø * | 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. |

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. After the directive is executed, the current line pointer will be positioned at the new position of the last line copied. Some lines may be renumbered after a copy.

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 /HELLO DOLLY/ +/END OF RANGE/ | After the next line which contains the string "HELLO DOLLY" place a copy of each line starting with the current line through the line containing "END OF 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 (and 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 (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. If renumbering occurs, then the message "SOME LINES RENUMBERED" will appear, indicating that renumbering took place.

EXAMPLES:

| | |
|---|---|
| INSERT | Accept line input after the current line. |
| I | |

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 numbers overlap line numbers previously in the file.

EXAMPLES:

I THIS BELOW THE CURRENT LINE OF THE FILE

INSERT EVERYTHING AFTER THE FIRST BLANK.

L[IST] [<target>]

MEANING:

LIST lines on the system printer through DOS68. Beginning with the current line, lines are printed on the system printer until the line specified by target is reached. By default, only the current line will be listed.

EXAMPLES:

| | |
|---|---|
| L | List the current line. |
| LIST 5 | List five lines starting with the current line. |
| L -1Ø | List the current line and the nine previous lines. |
| LIST *STRING* | List all lines down through the next line containing "STRING". |
| L -/STRING/ | List all lines up through the next previous line containing "STRING". |

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

The current line (and successive lines until the <range-target> is reached) is moved so that it follows 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 positioned at the new position of

the last line moved. Some lines may be renumbered after a MOVE.

EXAMPLES:

| | |
|---|---|
| MOVE 3 | Move the current line down three lines. |
| MO 1 /TARGET STRING/ | Insert the current line and all lines down through 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 "PROGRAM"/ |
| MO 10 -5 | Move the current line and the four previous lines below line 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 (default is 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 forms of the OVERLAY directive with these differences:: (1) The current line is not printed. (2) The remainder of the directive

EXAMPLES:

```
OVERLAY----AT--------(CURRENT)-----
```

25.00 = THAT IS THE (CURRENT) LINE

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 five lines starting with the current line. |
| P -10 | Print the current line and the nine previous lines. |
| PRINT *STRING* | Print all lines down through 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 ten 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 THE REPLACEMENT TEXT.

(null)

MEANING:

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

## SYSTEM CHARACTERISTICS

MAXIMUM LINE NUMBER - 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 Ø), therefore, the editor should not be used with files of 1Ø,ØØØ lines or longer. (this is not really a limitation since 1Ø,ØØØ null lines followed by a carriage return uses up to 4ØK of memory!

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

EDITOR SYSTEM INPUT/OUTPUT - All editor I/O, whether disk, terminal, or printer, is processed with the drivers used by DOS68. This is to point out that the terminal control parameters defined by the user through SET.$ are in effect during the operation of the editor. Please refer to the DOS68 documentation regarding the terminal control parameters and the definition of them with SET.$

## MODIFYING THE EDITOR

A partial source listing is included with this manual to aid those who would wish to modify the editor. Most modifications will not be required by most users, therefore only a few parameters will be elaborated upon here. Users who wish to modify the editor beyond the scope of the parameters mentioned here are urged to order a complete source listing of the editor.

To make changes in the editor, type GET,EDIT.$ followed by a carriage return. This will load the editor and return control back to DOS. After making the changes, use the SAVE command to save the editor back on the disk as a command file. Save from $0100 to $1D00, with a transfer address of $0100.

MEMORY END - The amount of memory used by the editor for its text buffer is determined primarily by the DOS68 MEMAX parameter. If for some reason MEMAX is undefined ($0000) then the editor parameter MEMTOP will be used. It is extremely important to note that both MEMAX and MEMTOP must not be lower than $2000, otherwise unpredictable results may occur. Refer to the DOS68 documentation regarding MEMAX and how to set this system parameter.

CURSOR CONTROL STRING - The cursor control string printed upon the receipt of the X command is located at location CNRSTR. It is currently set to 6 nulls, but may redefined by the user as required. The string must be terminated by $04.

PARTIAL SOURCE LISTING

```
0100            0095          ORG    $100
                0096
                0097 * PROGRAM STARTS HERE
                0098
0100 7E 163A    0099 START    JMP    DEDIT
0103 7E 0410    0100 RESTRT   JMP    PEDIT
0106 5E 00      0101 MEMTOP   FDB    MEMLIM       MEMORY LIMIT
                0102
                0103 * EXTERNAL I/O ROUTINES
                0104
0108 BD 72C4    0105 INCH     JSR    ZGETCH       CRT INPUT
010B 7D 0048    0106 OUTCH    TST    PRTFLG       HARD-COPY ON?
010E 26 03      0107          BNE    POUCH
0110 7E 72C1    0108          JMP    ZPUTCH       CRT OUTPUT
0113 7E 7312    0109 POUCH    JMP    ZHCOUT       PRINTER OUTPUT
0116 7E 730F    0110 PINIT    JMP    ZHCINT       PRINTER INIT
                0111 * USER DEFINABLE 'X' COMMAND CHARACTER STRING
0119 00         0112 CNRSTR   FCB    0,0,0,0,0,0,4
                0113
```

## MINI-TUTORIAL

The purpose of this section is to briefly introduce the reader to the use of the SSB 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 =SSB 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 =#
   15.00 =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 editor printed the last line and returned with its prompt (#). 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 form of an edit command is:

<line><directive>

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

```
#12 P
   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=#
#11I
   11.10=ANOTHER INSERTED LINE.
   11.20=#
#6P
    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 is deleted.

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 support 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 <targets> are: 3 (delete 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 <1 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

```
#150
#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

```
#D2
#
```

the editor will delete two lines starting with the current line. In our example, since we were at line 6.00, the "D2" 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=SSB 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 AAAA (AAAA on some terminals) and BBBB (1 on some terminals). The up arrow AAAA is used to designate the top or first line in the file. The down arrow BBBB 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
#+3P
    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 for 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 theh 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) AAAA (AAAA on some terminals) to denote the first line of the file, and (8) BBBB (BBBB on some terminals) to denote the last line of the file.

Now let's 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) AAAA to denote the top line in the file, and (6) BBBB 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=SSB 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=SSB 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=BBBBBBBBB
     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 became 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 current 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=SSB 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> /<string> / <target> <occurrence>

where "/" is a delimiter character to separate the two character strings; "<string> " is the character string to be replaced; "<string> " is the string of characters to replace them; "<target>" specifies the range of the changes; and "<occurrence>" specifies which occurrence(s) is 1 or is not specified, then only the first occurrence of <string> 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> 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.
#8.1 C /THIS IS //
    8.010=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 8.10. 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 different ways: STOP, S, OR LOG. This will

return you to your system monitor.

Now let's 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=SSB 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 $RE SOME NONSENSE LINES:
    5.25=THIS LINE CREATED WITH A"EQUALS".
    8.00=REPLACE CURRENT LINE HERE
    8.10=AN INSERTED LINE.
    8.20=SO IS THIS.
    9.00=THIS EDITOR IS FUN TO USE!
   10.00=BBBBBBBBBB
   11.00=
   11.10=ANOTHER INSERT?D LINE.
   12.00=THIS IS THE END OF THIS FIL?,
   13.00=AT LEAST FOR NOW.
#2C/EDITING/EDITOR/
    2.00=SSB 6800 TEXT EDITOR SYSTEM.  A NUMBER OF
#/BBB/
   10.00=BBBBBBBBBB
#-;THIS IS ; C 'E'XX' !
    1.00=THIS IS AN XXXAMPLE OF THE FANTASTICALLY USEFUL
    2.00=SSB 6800 TXXXT EDITING SYSTEM.  A NUMBER OF
    3.00=XXX$MPLES WILL BE SHOWN TO 70LOW 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 FUNE 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=BBBBBBBBB
#-1I
    9.10=TEST-TEST-TEST
    9.20=1234567890
    9.30=#D!
 BOTTOM OF FILE REACHED
#D!
 BOTTOM OF FILE REACHED
#1P!
#LOG
```

The previous tutorial has been only a brief introduction to the SSB Text Editing System. It is important to read and study the entire manual in order to fully understand all the power and features of this editor.