EDIT - DOS/65 EDITOR

VERSION 2.1

180 Ridge Road
Cimarron, CO 81220

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

1.1 OVERVIEW AND CONCEPT

The DOS/65 Editor, EDIT.COM, can be used to create or alter text files. The editor has context, character, and line oriented features and can edit files up to the capacity of the diskette.

The basic concept around which the editor is built is illustrated in Figure 1. All actual editing is done in a text buffer whose size is determined by the amount of memory available on the system. Files of any length can be edited regardless of memory size. That is possible since lines of text can be appended to the end of the text buffer from the source file and can be written to the destination buffer from the beginning of the text buffer once the necessary editing is finished. The memory size thus only limits the amount of text viewable at any instant, not the maximum file size.

Another key feature of EDIT is use of a cursor. The cursor is the pointer into the text buffer which marks the location from which editing commands are executed. In the example shown below, a portion of the text buffer is shown. The cursor is between the "h" and "e" of the word "the".

```
. . .and(cr)(lf)now is the time for all(cr)(lf)good. . .
                          ↑
```

Thus a command which would delete five characters to the right of the cursor (5D as we shall see later), would delete the string

```
'e tim'
```

from the buffer and leave the text buffer as follows:

```
. . .and(cr)(lf)now is the for all(cr)(lf)good. . .
                          ↑
```

1.2 KEY FACTS

It is important to note that:

- Each "line" includes a carriage return and a linefeed at the end. Each of these counts as a character in character oriented moves, deletes and searches.
- When positioned at the beginning of a line the cursor is between the linefeed of the previous line and the first character of the line.
- Directions from the cursor will normally be given as left or right. Left means towards the beginning of the text buffer and right means towards the end of the text buffer. These conventions are natural when one considers that the primary usage of EDIT is with English text which reads from left to right.
- Within this document sequences such as (cr) or (ctl-z) are used to represent the single character named within the parentheses. Parentheses are also used to denote

strings; e.g., (string1). In either case, the string or single character is actually what is entered at the keyboard. The parentheses are not actually entered.

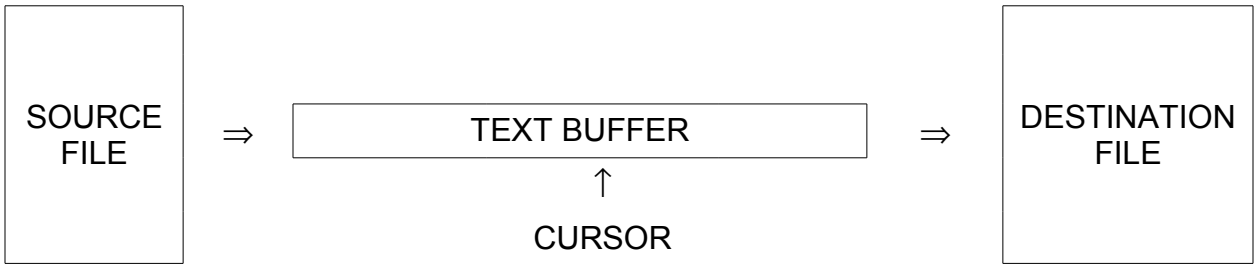


Figure 1. Editor Concept

SECTION 2 - EXECUTION

The editor is executed by use of a CCM command line such as the examples which follow (user inputs are underlined):

CCM INPUT	ACTION
A> edit source.asm	Edits file SOURCE.ASM on the default drive. Resulting file (SOURCE.ASM) is written to the default drive and the original file is unchanged except that its name is changed to SOURCE.BAK where BAK stands for backup.
A> edit b:source.asm	Same as first example except drive B is used instead of drive A. Note that EDIT.COM in this case is still loaded from drive A.
A> edit source.asm b:	Same as first example except that the resulting file is written to drive B. SOURCE.BAK would remain unaltered.
A> b:edit source.asm	Same as first example except that EDIT.COM is loaded from drive B

These examples are indicative of the flexibility available when invoking the editor. The command system can be generalized as follows:

```
(drive:)edit (drive:)ufn (drive:)
```

where the (drive:) terms represent optional items as discussed above. At initial invocation and during the editing session, the following error messages may be printed on the console:

DESTINATION FILE EXISTS
: A different destination drive is specified and the specified file exists on both the source and destination drive.

PEM FILE ERROR (FULL?)
: Other DOS related errors occur (e.g., the attempt to open the source file for read operations is unsuccessful).

All such errors result in an aborted run. The precise effect the abort has on the original source is a function of when the error was detected. In general the original file should be recoverable, however, other temporary work files (such as the output file) may not have been created or may not have been properly deleted, closed or renamed.

Note that if the source file does not exist a new file is created which is initially empty.

SECTION 3 - COMMANDS

3.1 BACKGROUND

A summary of the available commands is contained in Table 1. These commands fall into different categories. Each of these categories is discussed below with a detailed explanation of each command. These are some general features of the command syntax which should be understood first. Those features are discussed below.

Prompt

The editor prompts all command inputs with the character "*".

Numbers

Several commands can be preceded by a number (n). That number must be entered as a decimal number and can be preceded by a minus sign where permitted by the command in question. The legal range of numbers is -65535 to 65535. A shorthand way of entering 65535 is provided by use of the symbol #. If no number is entered, a number of +1 is assumed by the editor for those commands which can be preceded by a number. For some commands the number zero (0) has special meaning. Such special meanings will be discussed for each command.

Uppercase/Lowercase

The editor includes a case translation feature so that commands may be entered in upper or lower case. The editor, however, will not translate lower case characters into uppercase characters within a string or when inserting characters into the text buffer using the Insert command. Thus, the following commands are equivalent and will Find the first occurrence of the string "the".

```
Fthe       fthe
```

while the command

```
fTHE
```

will find the first occurrence of the string "THE".

Strings

For stand-alone, single string commands (such as those shown above) no special string termination character is needed. For the Substitute command and for combined commands it is necessary to use the eof character (ctl-z) in order to properly delimit the end of each string. Thus the following command:

```
Fthe(ctl-z)-1D
```

will find the first occurrence of the string "the" and will then delete one character towards the beginning of text buffer. The character deleted is the "e" as will be obvious after review of the Delete command. The point here is that string ends must be delimited with a (ctl-z) when any other characters follow the end of the string. For the Find command the string can be up to 128 characters long while for the Search command the combined length of the two strings is limited to 128 characters. (Also, see note below about total command line length.)

TABLE 1. COMMAND SUMMARY

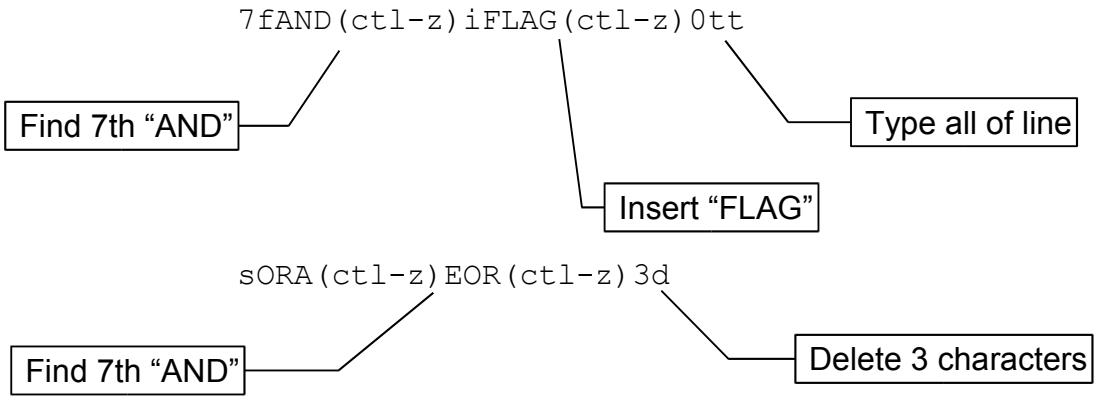
`+n`	Move n lines right (+) or left (-) in text buffer and type next line
`nA`	Append n lines to text buffer from source file
`+B`	Move cursor to Begin (+) or end (-) of text buffer
`+nC`	Move cursor n Characters right or left in text buffer
`+nD`	Delete n characters right (+) or left (-) of cursor
`E`	Normal Exit from editor
`nF(string)`	Find nth occurrence of string
`H`	Exit from editor and re-enter at Head of new source file
`I`	Insert text at cursor
`+nK`	Kill n lines right (+) or left (-) of cursor
`+nL`	Move cursor n lines right (+) or left (-) in text buffer
`nM`	Execute Macro n times
`O`	Restart editor with Original source file
`Q`	Quit with no file changes
`R`	Read library file into text buffer starting at cursor
`nS(string1)(ctl-z)(string2)`	Substitute string 2 for nth occurrence of string 1
`+nT`	Type n lines on console right (+) or left (-) of cursor
`nW`	Write n lines to destination file
`nX`	Trans(X)fer n lines to file X$$$$$$$$.LIB

NOTE

Text is not automatically moved from the source file to the text buffer at initial execution of EDIT.COM. The user must use the A (Append) command to move all or part of the source file to the text buffer before any editing can be accomplished.

Combined Commands

As implied by the preceding discussion, commands may be combined to achieve the desired editing effect. To combine commands the user simply enters them in the order desired when the prompt is displayed on the console. Thus in addition to the example shown above, the following are all legal command lines:



It is important to note that a combined line is executed only once and in strict left to right order. Provisions have, however, been made for multiple executions of a command line through use of the special Macro command. That feature will be discussed during the individual command discussions.

Command Editing

Input commands may be edited using the normal DOS/65 buffered input editing features prior to entering the terminating (cr). Command lines are limited to 128 characters.

Break

For those commands which involve multiple lines (e.g., a type command; such as, 150t) or a Macro command which is executed more than once, it is possible to terminate execution by typing any key.

CAUTION

Do not use (ctl-c) as execution might be terminated with unpredictable results.

The editor will then display a message of the form

```
BREAK - CAN NOT DO COMMAND SPECIFIED TIMES AT x
```

where x will be the command line character which the editor is currently processing. In many cases no character will be displayed if the command has processed up to the end of the command line.

In addition to user generated "breaks" the editor will generate similar messages when appropriate termination conditions are detected. Those conditions are:

Unrecognized Command

```
BREAK - UNRECOGNIZED COMMAND AT x
```

Text or String Buffer Full

```
BREAK - MEMORY BUFFER FULL AT x
```

Library File not Found

```
BREAK - LIBRARY FILE ERROR AT x
```

Cannot Execute Command Specified Number of Times (e.g., could not find string)

```
BREAK - CAN NOT DO COMMAND SPECIFIED TIMES AT x
```

3.2 COMMAND DESCRIPTIONS

3.2.1 GROUP 1 (EDITOR TERMINATION)

All commands within Group 1 must be entered as single commands without any preceding number or sign. They cannot be part of a combined command or a Macro command. These commands are:

E (Exit)

H (Head)

O (Original)

Q (Quit)

In addition to the requirements noted above the O and Q commands will be checked (verified) by issuing a line of the form

Q (Y/N)?

If the response is not a Y (upper or lower case) the command will be ignored.

3.2.1.1 E (EXIT)
The E command is the normal editor termination command. It will cause the text buffer contents (if any) to be transferred to the destination file and any unedited portion of the source file to be written to the destination file. The destination file is given the original source file name. The original source file is unaltered except that the type portion of its name is changed to .BAK for "BACKUP". Any X$$$$$$$.LIB file (see X command) created during the session is deleted. DOS/65 is then re-entered by executing a warm boot.

3.2.1.2 H (HEAD)
The H command operates exactly like the E command except that after the functions named in paragraph 3.2.1.1 are performed the editor is re-entered with the new source being the destination file resulting from the current editing session. This command is especially useful when several editing passes must be made through a file which is too big to fit in memory at one time.

3.2.1.3 O (ORIGINAL)
The O command restarts the editor with the original source file. All editing which has occurred during the session is ignored and any X$$$$$$$.LIB file created during the session is deleted.

3.2.1.4 Q (QUIT)
The Q command exits the editor with the original source file unaltered. The destination file is deleted as is any X$$$$$$$.LIB file created during the session.

3.2.2 GROUP 2 (LIBRARY FILE)
All commands within this group deal with files of type .LIB and involve insertion of existing or temporary files into the text buffer or creation of temporary library files.

3.2.2.1 nX (TRANSFER)
This command is used in conjunction with the R command to transfer blocks of text from one point to another. The X command transfers n lines to the right of the cursor location to a special temporary file named X$$$$$$$.LIB. This file will be written to the default drive. As explained in Section 3.2.1, the file will be deleted upon exit from the editor. N must be positive. If N=0, then any existing X$$$$$$$.LIB file is deleted and no new file is created.

If a X$$$$$$$.LIB file was created earlier in the session but not deleted, another nX command will automatically delete the previous temporary file. Note that the X command does not delete any text from the text buffer, it merely copies the text to the temporary file.

3.2.2.2 R (READ)
The R command will read data from a .LIB file into the text buffer beginning after the current position of the cursor. If the X$$$$$$$.LIB file is to be used, the R is used alone. If another file of type .LIB is to be used, the R must be followed by the name portion of the file. Thus the following are valid examples of the R command:

`R`	reads X$$$$$$$.LIB file
`Rtest`	reads TEST.LIB file

while the following are invalid examples of the R command:

`R*`	invalid file name
`Rabcdetghi`	file name too long

The R command cannot be preceded by a signed or unsigned number. It can be used within a combined command by using a (ctl-z) as a string terminator. Thus the following examples are valid.

`Rtest(ctl-z)7D`	reads file TEST.LIB then deletes seven characters after the added text
`R(ctl-z)fthe`	reads the file X$$$$$$$.LIB then finds the first occurrence of "the" after the added text

3.2.3 GROUP 3 (BUFFER TRANSFER)
The two commands in this group provide the only means of moving data to the text buffer from the source file and from the text buffer to the destination file. Both commands may be preceded by a positive number identifying the number of lines to be transferred. If no number is entered, then 1 is assumed. If 0 is entered, it is converted to 1.

3.2.3.1 nA (APPEND)
The A command will move n lines from the source file to the end of the text buffer. If all n lines cannot be transferred, a BREAK message will be generated. In that case, it is unlikely that an even number of lines was added to the text buffer. That should not be a problem as once space is opened-up in the text buffer through use of a W, K, D, or S command the rest of the incomplete line can be transferred by using a 1A or A command.

3.2.3.2 nW (WRITE)
The W command will move n lines from the beginning of the text buffer to the destination file. This command frees space in the text buffer and hence is useful when the available memory size is not sufficient to hold the entire file at one time.

3.2.4 GROUP 4 (LINE ORIENTED)

The commands in Group 4 act on the contents of the text buffer in terms of lines. These commands thus move, delete, or type whole lines rather than a certain number of characters and do not examine the contents of each line except to detect the (lf) which is the last character in each line. If the cursor is not positioned between the end of one line and the start of the next line when the command is executed, special action is taken for several of the commands. Such special features, if any, will be noted for each command.

3.2.4.1 +nL (LINES)

The L command is the simplest of all line oriented commands. Its execution results in the cursor being moved n lines left in the text buffer if n is negative. If n is positive, then the cursor is moved n lines right toward the end of the text buffer. If n is zero and the cursor is not positioned immediately prior to the start of a line, then it is moved left to the start of the line within which it is positioned. The cursor will not as a result of this command (or any other for that matter) be positioned outside the limits of the text buffer.

3.2.4.2 +B (BEGINNING)

The B command provides a means to rapidly move to either the beginning or end of the text buffer. A B (or +B) will move the cursor to the beginning of the text buffer while a -B will move to the end. The B command is in one sense nothing more than a shorthand way of entering a +#L command. In fact, the B command will produce the same results as the -#L command and the -B command will produce the same results as the #L command.

3.2.4.3 +nT (TYPE)

The T command does not move the cursor but it does provide a means to view all or part of the text buffer. If n is positive, then n lines right of the cursor towards the end of the text buffer will be typed. If n is negative, then n lines left of the cursor (i.e., towards the beginning of text buffer) are typed. In addition, if the cursor is not between the end of one line and the beginning of the next line and if n is negative, then that portion of the line which is left of the cursor will also be printed. As a special case for n=0 only that portion of the line which is left of the cursor will be printed. Thus the command 0tt will print the current line regardless of where the cursor is within the line.

3.2.4.4 +n

The use of a number (positive or negative) alone is a powerful and simple command. It is a shorthand way of entering a +nLT command. The effect of the command is to move n lines (left or right) and then type a single line to the right. Its most common use is realized by typing only a (cr) in response to the command prompt. Since the default value of n is +1, the effect of a (cr) only input is to move to the next line and type it. Because there is no command letter associated with this command it can only be used by itself or as the last individual command of a combined or Macro command. If n is zero, the cursor is moved left to the beginning of the current line and the line is then typed.

3.2.4.5 +nK (KILL)
The K command provides the ability to delete entire lines from the text buffer. If n is positive, then n lines to the right of the cursor are deleted. Similarly, if n is negative, then n lines left of the cursor are deleted. Similar to what happens with the T command when the cursor is not between two lines, a negative value of n will delete that portion of the line left of the cursor in addition to any whole lines deleted. If n is zero, only that portion of the line left of the cursor will be deleted.

3.2.5 GROUP 5 (CHARACTER ORIENTED)
The Group 5 commands are character oriented and thus move or delete characters. As pointed out in Section 1.2, the (cr) and (lf) are separate characters. When using character commands which involve a crossing of line boundaries, the user must ensure that both the (cr) and (lf) are properly counted.

3.2.5.1 +nC (CHARACTER)
The C command moves the cursor n characters left, if n is negative, or n characters right, if n is positive. If n is zero, no action is taken.

3.2.5.2 +nD (DELETE)
The D command deletes n characters to the left of the cursor if n is negative. If n is positive, then n characters to the right of the cursor are deleted.

3.2.6 GROUP 6 (STRING ORIENTED)
Perhaps the most generally useful commands are the string oriented commands. These commands allow the user to search for specific strings, to perform automatic search/replacement, or to insert arbitrary text into the buffer. Because of their usefulness and power several examples of these commands will be presented. Since the (cr) upon input will usually be interpreted as the end of a command line, a special character (ctl-1) has been designated as a substitute for the (cr)(lf) combination in string oriented commands. Examples of its use will be shown.

3.2.6.1 nF(string) (FIND)
This command will cause the editor to search for the nth occurrence of the specified string. If found, the cursor will be positioned immediately after the string. If it is not found, a BREAK message will be displayed on the console. When the F command is used by itself, the string need only be terminated by the normal (cr) used to terminate command input. When used in a combined command or in a Macro the EOF character (ctl-z) must be used. Thus, while the following two command lines are equivalent

```
3fstring(cr)

3fstring(ctl-z)(cr)
```

these two lines are not equivalent

```
2fstring-1d(cr)

2fstring(ctl-z)-1d(cr)
```

In the first case, both commands will search for the third occurrence of the string "string". In the second case, the first line will search for the second occurrence of the string "string-1d" while the second line will search for the second occurrence of the string "string" and, if found, will delete the "g" in "string".

The command line

```
fend(ctl-i)(ctl-l)
```

will search for the string "end(ctl-i)(cr)(lf)" and, if found, will position the cursor immediately after the (lf).

3.2.6.2 nS(string1)(ctl-z)(string2) (SUBSTITUTE)
The S command will search for the nth occurrence of string 1 and if found will replace it with string2. String2 need not be the same length as string1 and may be a null string in which case the effect is to delete string1. If x is the length of string 1 then the S command is equivalent to

```
nF(string1)(ctl-z)-xDI(string2)
```

The following are examples of typical S commands:

`SAND(ctl-z)ORA`	replaces AND with ORA
`S(ctl-i)(ctl-z)(ctl-l)`	replaces (ctl-i) i.e., a tab, with (cr)(lf)

3.2.6.3 I (INSERT)
The Insert command is unique in that it can operate in two modes. In the first mode, the I (or i) is followed by the string to be inserted into the text buffer and the string is terminated with a (ctl-z) or a (cr). In this mode, the string is inserted into the text and another command is requested with the command prompt "*". If the string is terminated with a (cr) a (cr)(lf) is also inserted as part of the string.

In the second mode, the I (or i) is followed immediately with a (cr). In this mode, characters will be entered into the text buffer until a (ctl-z) is entered. In this mode, the normal buffered input editing functions can be used (except that the (ctl-p) list enable toggle is inoperative and (ctl-c) is not recognized) to alter the text after it is entered. The key difference is that in this mode the entire text buffer has replaced the small (128 byte) buffer normally used for command line entry and the (cr) merely moves the cursor to the next line by automatically inserting and echoing the necessary (lf). It is possible to backspace to the left of a line and into the previous line. If that happens the contents of

the previous line will be printed on the console and it can then be altered. This process is rather inefficient but could conceivably be used to backup all the way to the beginning of the text buffer in order to correct a typing error. One reason for the inefficiency is that as characters and eventually lines are deleted or as lines are canceled, much of the text is lost. It would probably be better to terminate the insertion using the (ctl-z) and then use the other features of EDIT to correct the mistake without destroying all previous input. Examples of use of Insert command are shown in the Appendix.

3.2.7 GROUP 7 (MACRO)

There is a single Macro command (nM) which allows a single or combined command to be executed n times. Such a command is most useful for repetitive alterations of text such as:

```
mfASL(ctl-z)-3diROL(ctl-z)0tt
```

This particular Macro will change every occurrence of ASL to ROL and will type the complete line in which the change was made. The Macro command is unique in that if n=0 or n=1, then n is set to 65535 before execution. In other words, unless n is two or more it is assumed that execution will be attempted as many times as possible. Note that Macro commands cannot include the E, H, O, or Q commands.

As previously mentioned, Macro execution can be interrupted by pressing any key.

3.3 SUMMARY

The degree of flexibility afforded by the preceding commands is probably greater than will ever be fully exploited by any one user. Nevertheless, each and every feature of this editor is believed to be useful. The degree of utility of each command will be determined primarily by the cleverness of the user.