TYMSHARE MANUALS

REFERENCE SERIES

ADDENDUM TO EDITOR

JANUARY 1972

TYMSHARE, INC.
10340 BUBB ROAD
CUPERTINO, CALIFORNIA 95014

TYMSHARE®

ADDENDUM TO EDITOR

A new version of EDITOR has been released on all Tymshare 940 systems which extends the capabilities of this already versatile and powerful editing language. The new, expanded EDITOR provides the user with more power than ever before to modify programs, data, and text efficiently and easily.

In the new EDITOR, all characters are stored in 8-bit ASCII code, allowing the storage of both upper- and lower-case characters. The expanded text area can accommodate approximately 132,000 characters, or a maximum of 7,166 lines of text.

This addendum documents all of the new features added to the EDITOR language since the publication of the *Tymshare EDITOR Reference Manual*, dated July 1969, and supersedes all EDITOR addenda and bulletins published since that manual.

The new input and output commands include the capability to read and write ciphered files, to handle very large files, and to obtain paginated listings. Extended search features greatly facilitate beginning- and end-of-line searches and provide more power with the FIND command. The MARK command allows the user to mark certain lines for processing by subsequent commands. A new control character has been added to the editing repertoire. In addition, new features allow the joining and separating of individual lines.

Additional new EDITOR features allow the use of command files within the language itself, conditional execution of buffers, and direct entry into the BATCH FORTRAN compiler, FTC, with current EDITOR text.

In all examples in this document, everything typed by the user is underlined. Lower-case letters used in an example of a command form represent a group of letters that are typed. For example, the characters *file name* in a sample command form indicate that a legal file name should be typed at that point.

The symbols for user-typed Carriage Returns and Line Feeds are:

Carriage Return:	↵
Line Feed:	↴

Control characters are denoted by a superscript c. For example, A^c denotes Control A. The method of typing a control character depends upon the type of terminal being used. Consult the literature for the particular terminal being used.

INPUT AND OUTPUT FEATURES

Many new features have been added to the EDITOR language to facilitate input and output of text. The commands and features which concern specifically input or output are discussed later in this section.

The CIPHER command is used in both input and output. It allows the EDITOR user to read and write ciphered files. Such files are stored in a special code which cannot be deciphered without a special key, or password. This encoding provides an additional level of information security.

A file ciphered with the EXECUTIVE CIPHER program may be read in EDITOR by using the same key; similarly, a file ciphered in EDITOR may be unciphered with the CIPHER program.[1]

When the CIPHER command is given, EDITOR responds with the message:

KEY:

The user enters the key with which all subsequent file input and output operations with the READ, LOAD, WRITE, and SAVE commands are to be performed. This key is valid until the next CIPHER command is given. For additional security, the key is not printed on the terminal. Note that the CIPHER command cannot be used with the APPEND *file name* command or with append-only files.[1]

To stop ciphered input and output and to return to normal unciphered reading and writing, the CIPHER command is given, and the KEY: message is answered with a Carriage Return.

Example

```
*READ F1↵       The file F1 is read normally.
1918 CHARACTERS
*CIPHER↵
KEY:  ABC↵      The key does not actually print on
                the terminal.
*WRITE F1↵      File F1 is written as a ciphered file
 OLD FILE↵      with key ABC.
1848 CHARACTERS
*CLEAR↵
ALL? Y
```

1 - See the *Tymshare EXECUTIVE Reference Manual* for details on the CIPHER program and append-only files.

```
*READ F2↵            File F2 is read with key ABC.
3487 CHARACTERS
*CIPHER↵
KEY:↵                Ciphering ceases.
*WRITE F2UN↵         The contents of F2 are written
 NEW FILE↵           onto the unciphered file F2UN.
3370 CHARACTERS
*
```

Input Commands: LOAD and Range READ

Two input commands have been added to EDITOR: LOAD and range READ. The LOAD command is identical to the READ command. For example, the following are equivalent:

```
*LOAD INT↵                 *READ INT↵
442 CHARACTERS             442 CHARACTERS
*LOAD↵                     *READ↵
FROM: INTDATA↵             FROM: INTDATA↵
721 CHARACTERS             721 CHARACTERS
*                          *
```

The LOAD command has been added to make EDITOR compatible with other Tymshare languages in the manner of bringing files into the language. It does not replace the EDITOR buffer LOAD command, which requires a buffer number immediately preceding the command itself.

The user may manipulate very large files by reading and writing them one section at a time. A section of any file may be read into EDITOR, edited, and then written on a file. The text may be added at the end of the file or may be inserted at any point in the file.

The range READ command enters a section of a file into EDITOR. The forms of the command are:

***aREAD l_1,l_2 file name↵**

and

***aREAD l_1,l_2↵**
FROM: file name↵

where l_1 and l_2 are line numbers in the file, and a is an optional line address in the EDITOR text area. This command reads lines l_1 through l_2 and inserts them before the line addressed by a in the text area. No character count is printed. If a is omitted, lines l_1 through l_2 are appended to the existing text. Note that l_1 and l_2 are line numbers, not line labels. If l_2 is greater than the number of lines in the file, all lines from l_1 to the end of the file are read.

For example, the command

```
*5READ 75,981 AUDATA↵
```

reads lines 75 through 981 from the file AUDATA and inserts them before line 5 in the text area.

The second line in the line range may be a dollar sign ($) to indicate that the file is to be read from line l_1 to the end of the file.

The range READ command may be used to read as many as 7,166 lines or 132,000 characters.

The user may write a long file, one section at a time, by using the APPEND *file name* command.

Output Commands

Three new commands have been added to the EDITOR file output capability. These commands are APPEND *file name*, REPLACE, and SAVE. Terminal output has also been enhanced by three new commands: LIST, PAGE, and LINES.

A new message has been added in the QUIT command. If the user has changed his text but has not written it on a file before he gives the QUIT command, EDITOR prints:

```
FILE NOT WRITTEN, OK?
```

Answering Y returns the user to the EXECUTIVE. An N response returns the user to EDITOR command level, identified by the asterisk. He can then give any EDITOR command. No Carriage Return is necessary after the Y or N response.

The APPEND file name Command

The APPEND *file name* command can be used to append text in EDITOR to a specified file without destroying any of the existing text in the file. The form of the command is:

```
*rAPPEND file name↵
```

If the range r is omitted, the entire contents of the EDITOR text area are appended to the specified file.

When characters are appended to a file by the APPEND *file name* command, the new total character count of the file is printed. For example, BIG is a file containing 10,000 characters. The user appends 17,390 characters.

```
*APPEND BIG↵
• OLD FILE↵
27390 CHARACTERS       BIG now contains
*                      27,390 characters.
```

If there are no characters in EDITOR and the APPEND *file name* command is given, no character total is printed. Thus, the user is aware that he is appending no characters.

The Line Feed option may be used with the APPEND *file name* command to indicate that blanks are not to be compressed in the text appended to the file. For example,

```
*APPEND↓
TO: CUMFILE↵
 OLD FILE↵
922 CHARACTERS
*
```

Note that if the Line Feed option is used, the long form of the APPEND command must be given.

The following example demonstrates the use of the range READ and APPEND *file name* commands in manipulating a large file.

```
*READ 1,500 BIG↵
            The first 500 lines of the file BIG are read
            into EDITOR, and the desired substitutions
            are made.
*SUBSTITUTE↵
"COUNTD^c" FOR "NUMBERD^c"
WAIT? N
498
*WRITE BIGC↵     The revised text is written on a
 NEW FILE↵       new file, BIGC.
12396 CHARACTERS
*CLEAR↵    The EDITOR text area is cleared.
ALL? Y
*READ 501,1000 BIG↵
            The next 500 lines of the file BIG are read
            into EDITOR, and the desired substitutions
            are made.
*SUBSTITUTE↵
"COUNTD^c" FOR "NUMBERD^c"
WAIT? N
500
*APPEND BIGC↵      The revised text is appended
 OLD FILE↵         to the file BIGC.
23002 CHARACTERS    The file now contains
*                   23,002 characters.
```

The REPLACE Command

The REPLACE command allows the user to replace a section in a file with all or part of the text in EDITOR. The form of the command is:

```
*rREPLACE l1,l2 file name↵
```

Lines l_1 through l_2 in the named file are replaced by the EDITOR text specified in the line range r. Note that r may not be a single line number; however, if r is omitted, all of the text in EDITOR replaces lines l_1 through l_2 in the file. REPLACE must be used with ten lines or more; thus, l_2 minus l_1 must be greater than 9. For example,

```
*READ 200,300 BIGC↵
            Lines 200 through 300 of file BIGC are
            read into EDITOR.
*$=101      EDITOR contains 101 lines.
*SUBSTITUTE↵
"20D^c" FOR "70D^c"
WAIT? N
11
*FIND "67DATA" DELETE↵
WAIT? N
24          A total of 24 lines are deleted.
*$=77       EDITOR contains 77 lines.
*REPLACE 200,300 BIGC↵
            The contents of EDITOR replace lines
            200 through 300 of file BIGC. Note that
            the number of lines replaced need not
            equal the number of lines in EDITOR.
 OLD FILE↵
*
```

If the file named is a new file, the lines specified in the range r are written on the file, beginning at line 1 of the file. The parameters l_1 and l_2 are ignored in this case.

The SAVE Command

The SAVE command is equivalent to the WRITE command. It may be used with or without the address of a line or range of lines. For example, SAVE writes the entire contents of the text area onto a file. The rSAVE command writes the line or lines addressed by r on a file.

Like the WRITE command, SAVE has two options. When the SAVE command is followed by a Carriage Return, the text is written with multiple blanks compressed. SAVE followed by a Line Feed writes files with blanks uncompressed.

The following commands are identical:

```
*SAVE ACCT↵                 *WRITE ACCT↵
 OLD FILE↵                   OLD FILE↵
723 CHARACTERS              723 CHARACTERS
*                           *
```

The SAVE command has been added to EDITOR to provide compatibility with other Tymshare languages in the manner of writing files.

The LIST Command

In terms of action initiated, the LIST command is identical to the / command. The LIST command does, however, require a Carriage Return to execute the command. The LIST command thus permits the user to advance the terminal paper before typing a Carriage Return, thus providing a clean printout.

Like /, LIST has two forms. LIST prints the entire contents of the text area on the terminal. The rLIST command prints on the terminal the lines addressed by r.

If the LIST command is used without a line range, the current line is unchanged. If a range is used, the current line is set to the last line in the range.

Example

```
*LIST↵        LIST displays the entire text area.
THIS IS AN EXAMPLE
OF THE LIST COMMAND
WHICH IS IDENTICAL TO /
*2LIST↵       Line 2 is printed.
OF THE LIST COMMAND
*
```

The PAGE Command

The PAGE command is identical to the PRINT command except that the first page contains a heading at the top right-hand corner with the time and date. In addition, subsequent pages are numbered consecutively, in the upper right-hand corner, beginning with page 2.

The PAGE command may be followed by an integer number to indicate that the heading is to be omitted and that page numbering should begin on the first page with the specified page number. For example,

```
*PAGE 5↵
```

indicates that the first page contains no heading and is numbered as page 5. Subsequent pages are numbered consecutively, beginning with 6.

New Options With the LINES Command

The LINES command is normally followed by an integer to set the number of lines per page (excluding the top and bottom margins) for the PAGE and PRINT commands.

The LINES command may be used without a number to reset to 54 the number of lines per page. This command resets the number of lines per page to that produced if no LINES command had been given.

The LINES command may be followed by an equals sign (=) to determine the number of lines per page currently in use. For example,

```
-EDITOR↵
*LINES=54      The number of lines per page is 54
*LINES 25↵     when EDITOR is called.
*LINES=25      The new number of lines is 25.
*LINES↵
*LINES=54      The LINES command used alone sets
*              the number of lines per page to 54.
```

No form of the LINES command has any effect on the current line.

EDITING FEATURES

EDITOR's complete text editing capability has been enhanced with several new editing features. A new character, Control L, has been added to EDITOR's already extensive repertoire of editing characters. The JOIN command allows lines to be joined. In addition, EDITOR now contains easy methods for substituting Line Feeds and Carriage Returns for other characters and vice versa.

Control L

Control L is identical to Control H except that it does not print the old line. It copies the rest of the old line to the new line, printing only a plus sign (+) to indicate that editing may continue at the end of the line. The + does not become part of the new line. For example,

```
*5EDIT↵
SPECIAL ORDER 154
Lc+8↵
*5LIST↵
SPECIAL ORDER 1548
*
```

In the above example, Control L copies the rest of the old line to the new line, but not to the terminal, and prints the +. In this case, the entire line is copied. The edit continues at the end of the line where the user adds an 8 and terminates the edit with a Carriage Return.

The JOIN Command

The general form of the JOIN command is

```
*nJOIN↵
```

where n is a line number. This command joins lines n and n+1, replacing the Carriage Return at the end of line n with a Line Feed. Thus,

```
*17JOIN↵
```

joins lines 17 and 18 to form one EDITOR line, composed of two physical lines.

Each time a JOIN command is given, the number of lines in EDITOR is reduced by one. After the JOIN command is given, the current line is the line joined. For example, 15JOIN sets the current line to 15.

Example

```
*READ TEST↵
56 CHARACTERS
*LIST↵
THIS WAS LINE ONE
THIS WAS LINE TWO
THEY WILL BE JOINED
*$=3              There are three lines in the text area.
*1 JOIN↵
*1 LIST↵                   After the JOIN command,
THIS WAS LINE ONE          line 1 consists of two
THIS WAS LINE TWO          physical lines.
*$=2              There are now two lines in the text area.
*LIST↵
THIS WAS LINE ONE
THIS WAS LINE TWO
THEY WILL BE JOINED
*WRITE TEST↵
 OLD FILE↵
56 CHARACTERS
*
```

Several lines may be joined with the JOIN command, but the total number of characters, including Line Feeds and Carriage Returns, in the resultant line must not exceed 256. If this limit is exceeded, the error message

LINE TOO LONG

is printed, and the user is returned to EDITOR command level, indicated by the asterisk. No joining takes place in this case.

The SUBSTITUTE command, described below, can be used to replace any character except the Carriage Return. With the JOIN command, these Carriage Returns may be replaced with Line Feeds. The user may then make substitutions for the Line Feeds. Note that the total number of characters per line may not exceed 256.

Example

The user wishes to put the word STOP after each of the words ONE, TWO, and THREE on a single line with the word END. He uses the JOIN command three times, then the SUBSTITUTE command to substitute the word STOP for each Line Feed.

```
*/
ONE
TWO
THREE
FOUR
END
*1JOIN↵
*1JOIN↵
*1JOIN↵
*$=2          There are now two lines in the text area.
*1/
ONE
TWO           Line 1 consists of four physical lines.
THREE
FOUR
*SUBSTITUTE↵
" STOP D^c" FOR "↴   The word STOP is substi-
D^c"                  tuted for each Line Feed.
WAIT? N
3
*/
ONE STOP TWO STOP THREE STOP FOUR
END
*$=2
*
```

NOTE: $JOIN has no meaning. One line cannot be joined to a line which does not exist.

Substituting Line Feeds and Carriage Returns

With one exception the SUBSTITUTE command may be used to substitute Line Feeds and Carriage Returns for any characters in the text area and vice versa. The SUBSTITUTE command cannot be used to eliminate a Carriage Return. For example,

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

To substitute a Carriage Return for a character, the Carriage Return must be preceded by a Control V. The substituted Carriage Return is included as text and does not erase any characters. It does, however, create new lines. For example,

```
*/
FIRST:SECOND:THIRD
*$=1
*SUBSTITUTE↵       A Carriage Return is substituted
"V^c↵              for each colon in the text.
D^c" FOR ":D^c"
WAIT? N
2
*/
FIRST
SECOND
THIRD
*$=3        There are three lines in the text area.
*
```

The Carriage Return may also be included in the string to be replaced by preceding it with a Control V. This permits end-of-line substitutions. For example,

```
*/
THIS IS THE FIRST LINE
LINE 2
THIRD LINE
A LINE OF TEXT
*SUBSTITUTE↵             The user wishes to substitute
"TEXTV^c↵                TEXT for LINE only if LINE
D^c" FOR "LINEV^c↵       occurs at the end of the line.
D^c"
WAIT? N
2
*/
THIS IS THE FIRST TEXT
LINE 2
THIRD TEXT
A LINE OF TEXT
*
```

If a Control V followed by a Carriage Return is included in the text to be substituted at a location other than the final character, the text following the Carriage Return becomes a new line. For example,

```
*/
THIS IS LINE
2ND LINE
LINE 3
LAST
*$=4
*SUBSTITUTE↵
        The text END↵ NEW LINE is substituted for
        LINE only if LINE occurs at the end of the line.
"ENDV^c↵
NEW LINED^c" FOR "LINEV^c↵
D^c"
WAIT? N
2
*$=6            Two new lines are created by the
*/              text NEW LINE substituted twice.
THIS IS END
NEW LINE
2ND END
NEW LINE
LINE 3
LAST
*
```

NOTE: Carriage Returns may be inserted in text using buffers. In this case, however, the Carriage Return must be preceded by two Control V's. For example, line 1 consists of the text THIS IS ONE LINE. The user wishes to create two lines by inserting NOT↵ JUST between IS and ONE. For purposes of illustration, this is accomplished with a buffer, as follows.

```
*5LOAD↵
U^c&UE^c&E NOTV^cV^c&V↵
JUSTE^c&EF^c&FD^c
```

```
*5BUFFER↵
"&U&E NOT&V
"
"JUST&E&F"
*1EDIT↵
THIS IS ONE LINE
B^c#5

*1/
THIS IS NOT
*2/
JUST ONE LINE
*
```

SEARCHING FEATURES

The EDITOR language can be used to search the text area for the occurrence of particular sequences of characters. The lines found in such a search may be edited repetitively or marked for further EDITOR commands. This section describes a new text address capability, extended use of the FIND command, and two commands, MARK and UNMARK, which are used in marking specific EDITOR lines for future operations.

New Line Addressing

The EDITOR user may now address lines by using text at the beginning or end of the line. Such addressing is in addition to the other forms of searching, denoted by the colon, angle brackets, and quote marks.

A line may be addressed by text at the beginning of the line by surrounding the text by exclamation points. Such text must begin in position 1 of the line. For example,

```
*/
JOHNSON, E.A.
MASON, PAM
ROSS, JOHN
*!MA!/
MASON, PAM
*
```

Any of the following exclamation point addresses could also be used with the line addressed above:

```
!M!
!MASON!
!MASON,!
!MASON, PA!
```

The text used within the exclamation points must not contain an exclamation point.

Text at the end of a line may be used to address the line by surrounding such text with single or double quote marks and preceding the Carriage Return by a Control V. The Control V inhibits the normal function of the Carriage Return. For example,

```
*/
JOHNSON, E.A.
MASON, PAM
ROSS, JOHN
*"HNV^c↵          The user wishes to address a line
"/                ending with the characters HN.
ROSS, JOHN
*
```

Extended Use of the FIND Command

To allow the combination of search and edit capabilities, the FIND command may be used with any of the following EDITOR commands: APPEND *file name*, DELETE, EDIT, MODIFY, INSERT, MARK, UNMARK, PAGE, PRINT, REPLACE, SAVE, WRITE, SUBSTITUTE, GET, LOAD, LIST, /, =, and ←. For example,

```
*FIND "SFO" SUBSTITUTE↵
"#D^c" FOR "%D^c"
WAIT? N
8          Eight substitutions were made.
4          Four lines containing SFO were found.
*
```

The MARK and UNMARK Commands

The MARK command has the form

```
*rMARK n↵
```

where r is the address of a single line or range of lines, and n is the number 1 or 2. The range r must be included.

The MARK command marks all lines in the range and adds them to the MARK 1 or MARK 2 list. The lines marked may then be referred to by the address @1 or @2. For example, to print the lines in the MARK 2 list, the command

```
*@2/
```

is given.

Example

```
*/
THIS IS 1
LINE 2
LINE 3
FOUR
LINE FIVE
*2,4 MARK 1↵       Lines 2 through 4 are put
*@1/               into the MARK 1 list.
LINE 2
LINE 3
FOUR
*
```

The line lists associated with MARK are cumulative. When a line is marked, it remains in the appropriate MARK list until deleted from the EDITOR text area, removed from the MARK list with the UNMARK command, or until the MARK list is cleared with the @1 CLEAR or @2 CLEAR command.

The UNMARK command removes lines from the specified MARK list. No lines are removed from the text area with this command. The form of this command is

```
*rUNMARK n↵
```

where r and n are as used in the MARK command. The range r may not be omitted. Only the lines addressed by r are removed from the specified list. For example,

```
*/
SAMPLE
THIS
IS
RIGHT
LEFT
*2,5 MARK 2↵       Lines 2 through 5 are placed
*@2/               in the MARK 2 list.
THIS
IS
RIGHT
LEFT
*4 UNMARK 2↵       Line 4 is removed from the
*@2/               MARK 2 list.
THIS
IS
LEFT
*
```

The EDITOR line number of a line may change after the line has been marked, for example, by deletion of a line above it. In this case, the line remains in its MARK list until removed by using the UNMARK command with its new line number. For example,

```
*/
ONE
TWO
THREE
FOUR
FIVE
SIX
*4,6 MARK 1↵       Lines 4 through 6 are put into
*@1/               the MARK 1 list.
FOUR
FIVE
SIX
*2 DELETE↵         Line 2 is deleted from the text
*/                 area, changing the numbers
ONE                of lines 4 through 6 to 3
THREE              through 5.
FOUR
FIVE
SIX
*3 UNMARK 1↵       Line 3, previously line 4, is
*@1/               removed from the MARK 1
FIVE               list but not from the text area.
SIX
*
```

The FIND command may also be used to insert and remove lines from the MARK lists. For example,

```
*/
EACH LINE CONTAINING AN M
WILL BE PUT INTO THE MARK
LIST.  THOSE CONTAINING A K
WILL BE REMOVED WITH UNMARK
*FIND "M" MARK 2↵      Three lines containing
3                      the letter M are put into
*@2/                   the MARK 2 list.
EACH LINE CONTAINING AN M
WILL BE PUT INTO THE MARK
WILL BE REMOVED WITH UNMARK
*FIND "K" UNMARK 2↵
        All lines containing the letter K are removed
        from the MARK 2 list.  Three lines contain
        the letter K, but only two are in the
        MARK 2 list.
3
*@2/
EACH LINE CONTAINING AN M
*
```

The CLEAR command may be used to clear an entire MARK list. For example,

```
*/
40 DATA 2.5,7.11
160 READ S1
170 DATA 457,899,0
180 DATA 468,999
*2 MARK 1↵
*@1/                The MARK 1 list is printed.
160 READ S1         Note that the lines printed
*:40: MARK 1↵       appear in EDITOR order,
*@1/                not in order of entry into
40 DATA 2.5,7.11    the MARK 1 list.
160 READ S1
*@1 CLEAR↵          The MARK 1 list is cleared.
*@1/
*
```

The lines marked by MARK remain in the same order as they appeared in the EDITOR text area. If line 1 had been marked before line 2 in the example above, the MARK list would still be in the order shown.

The addresses @1 and @2 may be used instead of the range r in the following commands:

rAPPEND file name	rPAGE
rDELETE	rPRINT
rEDIT	rREPLACE l_1,l_2 file name
rFIND	rSAVE
r;nGET	rSUBSTITUTE
r;nLOAD	rUNMARK
rMARK	rWRITE
rMODIFY	r/

In addition, a MARK address may be used with the commands INSERT, =, and ←. For example,

```
*@2 INSERT↵
*@1 =
```

and

```
*@2 ←
```

are all valid commands.

In all cases, the commands above work identically with @1 and @2 addresses as with the FIND command.

UTILITY COMMANDS

Four new utility commands have been added to the EDITOR language: COMMAND, FTC, ONRING, and VERSION. These commands are described below.

Command Files

Command files can now be executed within EDITOR by typing

```
*COMMAND file name↵
```

where file name is the name of the file on which the commands are stored.

Any command can be included in the command file; however, terminal output is suppressed in EDITOR, except in those cases when it is specifically requested, for example, with the /, PRINT, PAGE, and LIST commands.

Consider the following command file:

```
READ F1         Files F1, F2, and F3 are to be read
READ F2         into EDITOR.
READ F3
1,$-1 FIND 'END' DELETE
                All lines containing END are to be
                deleted except the last line.
N               N is the response to WAIT?.
/               The revised text is to be printed.
WRITE F123      The revised text is to be written on
                the file F123.
↵               The Carriage Return is the response
                to the OLD FILE/NEW FILE
                message.
COMMAND T       Subsequent commands are to be
                taken from the terminal.
```

These commands are written on the file CFIL, and the EDITOR text area is cleared. The CLEAR command must be used before the command file is executed if the command file was just entered into EDITOR; otherwise, the command file commands are interpreted as text. The procedure is illustrated below.

```
-EDITOR↵
*APPEND↵        The command file is created in ED-
READ F1↵        ITOR with the APPEND command.
READ F2↵
READ F3↵
1,$-1 FIND 'END' DELETE↵
N↵
/↵
WRITE F123↵
↵
COMMAND T↵      COMMAND T is the last line in
D^C             the command file. The APPEND
                command is terminated with a
                Control D.
```

```
*WRITE↵
TO: CFIL↵
 NEW FILE↵
74 CHARACTERS
*CLEAR↵                The EDITOR text area is cleared.
ALL? Y
*
```

The response to the / command is printed, but the responses to the READ, WRITE, and FIND commands are not printed, unless a command used with FIND requests terminal output. Responses are not printed for the LOAD, SAVE, and APPEND *file name* commands.

The commands that may be included in the command file are not limited to EDITOR commands. In the following example, EDITOR is exited and the command file CFL, called into EDITOR, executes commands until control is returned to the terminal with a COMMAND T command. CFL contains the following commands:

READ A1 **READ A2**	*Files A1 and A2 are to be read into EDITOR.*
1,$-1 FIND 'END' DELETE	*All lines containing END are to be deleted except for the last line.*
N	*N is the response to WAIT?.*
WRITE APROG	*The revised text is to be written on the file APROG.*
↵	*The Carriage Return response to the OLD FILE/NEW FILE message must be included.*
QUIT	*EDITOR is to be exited and SUPER FORTRAN called.*
SFORTRAN	
LOAD APROG **FAST**	*APROG is to be loaded and listed.*
QUIT	*The QUIT command returns to the EXECUTIVE.*
COMMAND T	*Control is returned to the terminal.*

Creation and execution of the command file occurs as follows:

```
*APPEND↵
READ F1↵
READ F2↵
1,$-1 FIND 'END' DELETE↵
N↵
WRITE APROG↵
↵
QUIT↵
SFORTRAN↵
LOAD APROG↵
FAST↵
QUIT↵
COMMAND T↵
D^c
*WRITE↵
TO: CFL↵
 NEW FILE↵
100 CHARACTERS
*CLEAR↵      The CLEAR command deletes the text
ALL? Y       of CFL from EDITOR.
*COMMAND CFL↵  The command file is executed.

10 A=23
20 A=A↑3
30 DISPLAY A
100 B=A↑2/3.
200 DISPLAY A+B
300 END

_
```

COMMAND has a long form also. The user may type

```
*COMMAND↵
```

after which EDITOR responds with:

```
FROM:
```

The user enters the name of the command file followed by a Carriage Return.

The long and short forms of COMMAND are equivalent.

The FTC Command

The FTC command provides more convenient BATCH FORTRAN compilations by allowing the user to enter the BATCH FORTRAN compiler with all the current EDITOR text without first writing the text on a file. The FTC command dismisses EDITOR and calls FTC, the BATCH FORTRAN compiler. The compiler accepts as input the text from the EDITOR text area. The user simply defines his output file and other options before compilation.

When FTC has been called from EDITOR, the user always returns to EDITOR if the exit is made by giving the BATCH FORTRAN QUIT command. An Alt Mode/Escape in FTC returns the user to FTC command level.

The EDITOR FTC command is demonstrated below.

```
*APPEND↵
XI^c    THIS IS A COMMENT↵
```

The program entered in EDITOR has an error; there should be a C and not an X in column 1 of the first line.

```
I^c     OPEN (5,INPUT,DATA)↵
I^c     ASSIGN 55 TO K↵
I^c     CALL EOF(K)↵
I^c     B=0↵
2I^c    READ(5,3) A↵
I^c     B=B+A↵
I^c     GO TO 2↵
55I^c   WRITE(1,4) B↵
3I^c    FORMAT(F9.0)↵
4I^c    FORMAT($ THE SUM IS $,F10.3/)↵
I^c     END↵
D^c
*FTC↵
```

The FTC command enters the BATCH FORTRAN compiler immediately.

```
+OUTPUT TO PRGC↵
 NEW FILE↵

+COMPILE↵

X       THIS IS A COMMENT
↑
SYNTAX
        END
```

The program error is identified by the compiler.

```
+QUIT↵
```

The user types QUIT to return to EDITOR. He may type EDITOR to return also.

```
*1MODIFY↵
```

The program is still in EDITOR and can be modified directly. Control D copies the remainder of the line after the initial character is corrected.

```
CD^c    THIS IS A COMMENT
*FTC↵
```

The user calls FTC again to recompile the program.

```
+OUTPUT TO PRGC↵
 OLD FILE↵

+COMPILE↵
```

No compilation errors are found.

```
        END

+QUIT↵
```

QUIT returns the user to EDITOR.

```
*QUIT↵
FILE NOT WRITTEN,OK? N
```

EDITOR warns the user that he has not saved the text on a file.

```
*WRITE PRG↵
 NEW FILE↵
185 CHARACTERS
*QUIT↵

-
```

Conditional Execution of a Buffer

The ONRING command allows conditional execution of a buffer. The command is of the form

```
*ONRING n↵
```

where n is the number of a buffer. The ONRING command determines that in succeeding buffer operations, a bell condition caused by any buffer command automatically transfers control to the buffer specified in the ONRING command.

A bell condition is said to occur when EDITOR is instructed to perform an impossible operation. The bell on the terminal does not necessarily ring when this occurs. For example,

```
*5EDIT↵
THIS LINE CAUSES A BELL CONDITION.
Z^cF
```

The letter F does not appear in the line. Thus, the request to copy up to and including the F causes a bell condition.

If the following commands are given, buffer number 1 will be executed if and when some operation in buffer 3 causes a bell condition.

```
*ONRING 1↵
*B^c#3
```

The ONRING command should be the last command executed before calling the buffer.

Assume that the following text is in EDITOR,

```
NOV   27.30 INT 55
JAN   76.32 INT 1.53
MAR  101.50 INT 2.03
JUN   50.36 INT 1.06
AUG   33.50 INT 67
```

and the user wishes to eliminate the decimal points in the numbers following INT in each line. He creates a buffer to copy each line up to the letter I, then to copy up to and delete the next decimal point in the line. The process is to be repeated for each line. The buffer to perform this task is created as follows:

```
*4LOAD ↵
Z^c&ZIO^c&O.S^c&SF^c&FB^c&B4D^c
*
```

A decimal point will not be found after the letter I in lines 1 and 5 of the text, and executing buffer 4 with with these lines causes a bell condition. The user creates a second buffer to be executed whenever a bell condition occurs, as follows:

```
*2LOAD ↵
F^c&FB^c&B4D^c
*
```

Buffer 2 simply copies the rest of the current line and returns control to buffer 4.

The operation can now be accomplished:

```
*ONRING 2 ↵
*1,$ MODIFY ↵
B^c#4
*/
NOV   27.30  INT  55
JAN   76.32  INT  153
MAR  101.50  INT  203
JUN   50.36  INT  106
AUG   33.50  INT  67
*
```

The VERSION Command

The VERSION command prints the number of the current version of EDITOR. It may be given at any time at EDITOR command level, denoted by the asterisk.