ECCE

the

Edinburgh Compatible Context Editor

Contents

Introduction

ECCE - the Edinburgh Compatible Editor - is a program for creating, updating, and extending text files. The design of ECCE is based on ideas developed by Alan Freeman, Chris Whitfield, and Hamish Dewar. Numerous versions of ECCE have been implemented on computers in the Computer Science Department and Regional Computing Centre at the University of Edinburgh. This document is adapted from one describing the 2900 EMAS implementation of ECCE. It describes a version of ECCE available on CDC NOS/BE at the University of London Computer Centre (ULCC). This was originally written in Pascal for the University of Manchester Regional Computing Centre (UMRCC) NOS system by Atholl Hay. It was adapted for NOS/BE and MVS by Andy Inston.

1. Calling the editor

1.1. From NOS/BE

The editor can be used to manipulate any standard NOS/BE character file. Assuming that the user wishes to call the editor as EC, first ... ATTACH, EC, ECCE, ID = TRIAL.

COMMAND: EC	-	this produces a new file called CURRENT or overwrites an existing file with that name.
COMMAND: EC, TEST	-	edits an existing file called TEST. (the editor will prompt the user for an output file name on completion of editing).
COMMAND: EC, TEST, NUTEST	-	this creates a file NUTEST (or overwrites an existing file of that name) which wil contain the result of the editing operation on TEST. File TEST remains unaltered.

The user is warned of the low value of default CP time on Intercomm. The ETL command should be used to overcome this.

The full set of available editor parameters is rather complicated -

COMMAND: EC [positional filenames][keyword filenames][/X][commands]

where the filenames are -

position	keyword	file	default value
1	E =	file to be edited	current
2	N =	result file	<none>
3	S =	secondary input file	
4	I =	command file	input
5	L =	list file	output

Note that if the result file is specified as *, then the file to be edited will be overwritten.

The parameter /X will suppress message generation by the editor.

Thew user may append the edit commands on the same lines as the editor call. In this case the effect of several command lines can be produced by inserting semicolons between the command groups.

1.2. From MVS

ECCE must be incorporated into your TSO CLIST.

The call to ECCE then has the following syntax -

ECCE {edit file|*} {0|result file|*|/} [S(secondary file)]

[I(command file)] [L(listing file)]

[CLIST|TEXT|DATA|FORT|CNTL|LIST|OULIST|LINKLIST|LOADLIST]

[NUM|NONUM]

[LINE({|rec||255}|FIXED|DCB('attributes')]

[C('command string')] [X]

[DEBUG] [VER({ecce binary ds|ECCE})]

Where at least the edit file or result file must be present but all parameters between [] are optional.

2. Commands

After invoking ECCE, commands are issued from a console, one to a line or many to a line. Spaces have no significance, except within text strings, and command lines are terminated by a newline. After reading the command line the editor checks it for syntactic correctness and then executes the individual commands in left to right order. A syntax error in a command line (such as an unmatched string delimiter) causes the whole command line to be ignored and an error report to be produced (see paragraph 8. Command failure).

To terminate the editing session the command &C is issued (&C must be the only command in the command line).

3. Position within a file - the file pointer

Editing commands (except the 'special' commands such as &C) operate on the source text at the current position in the text. In order to define the current position, the editor maintains a pointer henceforth called the file pointer. Conceptually this is always between two characters of the source text, or immediately before the first character of the file, or immediately after the last character in the file. When printing out a line at the console the editor identifies the file pointer by typing an up-arrow or caret character in the appropriate position unless this is at the beginning of a line (precisely which character is used to represent the file pointer depends on the physical characteristics of the console).

For example

```
THIS IS AN UP-AR^ROW OR CARET CHARACTER.
BUT HERE THE FILE POINTER IS AT THE START OF THE LINE.
```

For convenience, the end of the file is treated as an extra line following the last line in the file. This extra line is typed out as **END**. For example:

```
THIS IS THE LAST LINE OF MYFILE.
**END**
```

Similarly, an end-of-section marker appears as &EOS (and may be inserted as such). By default ECCE types out the current line after the execution of each command line, unless two conditions are true. Firstly the line must have been typed out in response to the immediately preceding command line, and secondly the file pointer must not have been moved (forwards or backwards) across a line boundary by the latest command line.

4. A simple subset of commands

In this section a simple subset of ECCE's facilities will be described. This is sufficient to perform most editing operations. More sophisticated commands and programmed commands will be described later. The commands described alow the file pointer to be moved forwards and backwards in the source text, a line of source text to be killed (removed), a line of text to be inserted into the file, occurrences of a specified text string to be found, lines of text to be printed, and the editing session closed.

All these commands (except &C) can be followed by a positive integer, zero (0), or *, to specify how many times the command is to be repeated. Zero and * denote indefinite repetition, i.e. the command is executed repeatedly until it fails or until a large number of repetitions (currently 5000) have been performed.

&C — Close the editing session. &C must be the only command in the command line.

M — Move the file pointer to the start of the next line. This fails if the file pointer is already at the end of the file (that is, after the last line of the file).

M- — Move the file pointer back to the start of the previous line. This fails if the current line is the first line of the file.

M£ — Move cursor to numbered line.

K — Kill the current line. That is, remove the whole of the current line from the file. The file pointer is left at the beginning of the next line (as for M). This command fails if the file pointer is already at the end of the file.

K£ — Kill lines to numbered line.

G — Get a file of text from the console and insert it between the end of the previous line (if any) and the start of the current line. The file pointer is left at the start of the current line (that is immediately after the new text). The user is prompted for input with a colon (:) and the command will fail if the first character of the input line begins with a colon, in which case the file pointer is not moved. The rest of the line after the colon (if present) is treated as a command line. (The GET prompt may be changed, see below).

F/TEXT/ Find TEXT. Search the rest of the file, starting at the file pointer, for the first occurrence of the string TEXT. A string is any sequence of characters (excluding newline) enclosed in quotes. Symbols other than those which have a defined significance in the command language must be chosen to represent the quotation mark, for example / " or . The file pointer is left immediately before the first occurrence of TEXT if TEXT is found. Otherwise the command fails and the file pointer is left at the end of the file. If the command is repeated, then the occurrence of TEXT just found is ignored in the subsequent search for TEXT (see paragraph 6. Text location and manipulation commands).

&A Abort the edit session. No output file is created.

P Print the current line at the console. If the multiple form of P is used, for example P6 or P*, then a move (M) is performed after the first and subsequent P's (but not after the final one). In this way a sequence of lines may conveniently be printed at the console. The simpke command P can never fail and does not move the file pointer. The multiple form of the Print command (for example P10) fails only if an implicit attempt is made to move beyond the end of the file (for example, trying to print 10 lines when there are only 8 lines before the end of file), and always leaves the file pointer at the start of the last line printed (or at the end of the file).

The P command identifies the file pointer by typing a caret or up-arrow character in the appropriate position, unless this is at the start of a line. The end of the file is identified as **END**. It should be noted that the multiple form of P is not strictly repetition, for the effect of P followed by P is to print the current line twice, whereas the effect of P2 is to print the current line and the next line, and move the file pointer to the start of the next line. Generally this does not cause any problems or confusion.

P£ Print lines to numbered line (should always be ahead of the current line)

C Move cursor to specified column in the line.

H Hop cursor to specified column in the line. The text to the right of the cursor is moved across with the cursor.

5. An example of the use of ECCE

In this section ECCE is used interactively to create and update a file of text, which will contain part of a poem by Roger McGough called "Discretion". The complete dialogue between ECCE and the user is displayed. The user is prompted for command lines with a '>' character and for input with a ':'. Starred lines are ECCE's normal (default) monitoring output (see paragraph 11. Monitoring commands). Note that '*' is not printed at the console, and that '>' and ':' are not inserted into the source text. Explanatory comments occur in the right hand half of the line starting with '/' character.

```
COMMAND:-EC,POEM                    /invoke the editor
                                    /to make a new file
>g*                                 /get some lines of input
:Discretion
:Discretion is the better part of Valarie
:(though all of her is nice)
:Lips as warm as strawberries
:eyws as cold as ice
:the very best of everything
:only will suffice
:not for her potatoes
:and pudding made of rice
:
:not for hwr potatoes
:and pudding made of rice
:she takes carbohydrates like God takes advice
::                                  /end the input for now

>m-9                                /move back 9 lines
* EYWS AS COLD AS ICE
>k g                                /kill it and get a replacement
:eyes as cold as ice
>f/pudd/                            /missed the 's' off the end
* AND ^PUDDING MADE OF RICE
>k                                  /delete the line
*                                   /current line after deletion
>g                                  /get a replacement
:and puddings made of rice
>f/hwr/                             /welsh boyo?
* NOT FOR ^HWR POTATOES
>k g                                /kill it, and replace it
:not for her potatoes
>m*                                 /move to end of file
**END**
>g*                                 /and continue the input ...
>g*
: ...
::
&C
RESULT FILE> NEWLFN
```

Note tha "NEWLFN" can now be CATALOGUED, a preceding REQUEST, NEWLFN, *PF B not required.

6. Text location and manipulation commands

It is apparent from the above example that the use of only the very limited subset of commands introduced so far is clumsy, tedious, and error prone. The commands described below can be used to move the file pointer to immediately before or after a specified text string, and to insert, substitute, or delete specified strings of text, rather than whole lines.

In may of the commands introduced in this section a search for a text string is implied. This search always begins at the character immediately following the file pointer and fails, with the exception of Find, if the end of the line is encountered before the text is fould. Find fails only at the end of the file.

In successive searches for a text string by Find or Uncover, the occurrence just found is ignored. By contrast, the Delete command will operate on the occurrence of text just found.

In all the following examples the "/" character is used as a string delimiter. Characters with no other significance to the editor should be used, for example ' or " or £ or . (or / itself).

F/TEXT/ Find TEXT. Search the rest of the file, starting at the file pointer for the first occurrence of TEXT. The file pointer is left immediately before the first occurrence of TEXT if it is found. Otherwise the command fails and the file pointer is left at the end of the file. If the previous command was a text location command (F, V, or U, see below) then the occurrence of TEXT just located is ignored when searching for TEXT. This applies particularly to the case of F's being repeated (for example, F/TEXT/5 which finds the 5th occurrence of TEXT).

S/STR/ Substitute STR for TEXT. If the previous command was F/TEXT/, V/TEXT/ or U/TEXT/ (see below) then delete TEXT and insert STR. The file pointer is left immediately to the right of STR. The command fails if the previous command was not an F, V, or U. Note that STR may be the null string, so that F/ABCD/S// effectively deletes the first occurrence of ABCD.

T/TEXT/ Traverse TEXT. Search the current line, starting at the file pointer, for the first occurrence of TEXT and move the file pointer to immediately after it. If TEXT is not found on the current line the command fails. In this case the file pointer is not moved. Traverse is rather like Find except that the file pointer is left after the occurrence of TEXT, but it is not a a text location command (Substitute cannot be used after it). Traverse is useful in many circumstances, for example when adding an 's' (or any other ending) to the end of a word.

D/TEXT/ Delete TEXT. Search the current line, starting at the file pointer, for the first occurrence of TEXT then delete it. The command fails if TEXT is not found before the end of the current line. If the command succeeds then the file pointer is moved to the position previously occupied by TEXT, otherwise the file pointer is not moved.

I/TEXT/ Insert the specified TEXT immediately before the file poitner. The file pointer is left immediately after the inserted TEXT. Insert will fail if the line length is already greater than 160 characters. If Insert fails then the file pointer is not moved.

U/TEXT/ Uncover TEXT. Search the current line, starting at the file pointer, for the first occurrence of TEXT and remove all characters between the file pointer and the start of TEXT (TEXT itself if not removed). The file pointer is left immediately to the left of TEXT. If TEXT is not found on the current line then the command fails and the file pointer is not moved. Otherwise TEXT may be replaced by using the S/STR/ (Substitute) command.

V/TEXT/ Verify that TEXT occurs immediately to the right of the file pointer. Fail otherwise. If successful, S/STR/ can be used to to substitute STR for the TEXT just verified (see above for S). This command is of use in programmed commands (see paragraph 14. Programmed commands).

The following combination of commands are often found to be useful:

```
F/TEXT1/ S/TEXT2/     /applies to rest of file.
D/TEXT1/ I/TEXT2/     /applies to rest of line.
T/word/ I/ending/
U/./                  /delete rest of sentence.
U/./S/,/              /and change full stop to comma.
```

7. A further example of the use of ECCE

In the earlier example of the use of ECCE only a very basic subset of commands was used. In this section the same example is reworked to show how the text location and manipulation commands, just introduced, can be used to make the necessary changes more easily.

The command and monitoring conventions, and the command and input prompts, are exactly the same as in the previous example.

```
COMMAND: EC                              /invoke the editor
ECCE <version>                           /to make a new file
>g*                                      /get some lines of input
:Discretion
:Discretion is the better part of Valarie
:(though all of her is nice)
:Lips are warm as strawberries
:eyws as cold as ice
:the very best of everything
:only will suffice
:not for her potatoes
:and pudding made of rice
:
:not for hwr potatoes
:and puddings made of rice
:she takes carbohydrates like God takes advice
:a surfeit of ambition
:is her particular vice
:Valarie fondles lovers
:like a mousetrap fonles mice
::                                       /end the input for now

>m-o                                     /move back to start
*DISCRETION
>f/eyws/ s/eyes/                         /correct first mistake
*EYE^S AS COLD AS ICE
>f/pudding/t/g/                          /missed the 's' off the end
*AND PUDDING^ MADE OF RICE
>i.s.                                    /so put the 's' back in
>f/hwr/ d/w/                             /remove the mistake
*NOT FOR H^R POTATOES
>i/e/                                    /and correct it
>f/y/                                    /look for next blunder
*SHE TAKES CARBOH^YDRATES LIKE GOD TAKES ADVICE
                                         /not there yet!
>f/y/                                    /so repeat the command
*IS HER PARTIC^YLAR VICE
>d/y/ i/u/                               /fix the error
                                         /N.D.  D is not a text location
                                         command
                                         /so 'y' just Found is deleted
>f/nle/ s/ndle
*LIKE A MOUSETRAP FONDLE^S MICE
>&C                                      /end the edit
```

```
RESULT FILE>POEM
EDIT OK
COMMAND-                          /next command prompt from Intercomm
```

8. Command failure

A command can fail in either of two ways. Firstly it may be syntactically incorrect in which case the whole command line is ignored and an error report produced: secondly it may fail in execution in which case a failure report is produced, the current line (at time of failure) is printed, and the rest of the command line is ignored.

In general the failure of a simple command leaves the file pointer unmoved; however if the failing command is part of a command line or repeated command then the file pointer is left positioned by the last successfully executed command.

Syntax errors include commands that are not recognised, mismatched string delimiters, mismatched parentheses (see paragraph 14. Programmed commands), command line size exceeded, and so forth.
For example

```
>w
 W?                    /unknown command
>f/hello.
 TEXT FOR F?           /mismatched string delimiters
>mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
 SIZE?                 /command too long
>g/abc/
G /?                   /wrong syntax entirely
```

Examples of execution failure might be

```
      **END**          /at end of file
      >m               /can't, so it will fail
      FAILURE: M
      **END**
```

and

```
      >m               /move to next line
      How now brown cow.
      >s/horse/ m      /meaningless!
      FAILURE: S'horse'
      How now brown cow.
                       /note, Move not executed
```

or

```
      >p               /print current line
      How now brown cow.
      >t/now/2         /Note that 'now' occurs only once
      FAILURE: T'now'
      How now brown cow.
                       /2nd T/now/ not possible
```

or

```
      >g3              /get 3 lines
      ::               /didn't mean it, so get out of G
      FAILURE: G
      ... current line ...
                       /echo of current line on error
```

Note that indefinitely repeated commands such as g* or m* produce no failure reports, but that the failure condition is used to terminate the repetition.

9. Character manipulation commands

Of the commands met so far M and M- move the file pointer past a whole line and K deletes a whole line. The next four commands operate on individual characters within a line.

R Right shift the file pointer one character position. This fails if the file pointer is already at the end of the current line

L Left shift the file pointer one character position. This fails if the file pointer is already at the beginning of the current line.

E Erase the character immediately to the right of the file pointer. This fails if the file pointer is already at the end of the current line.

E- Erase the character immediately to the left of the file pointer. This fails if the file pointer is already at the beginning of the current line

It is useful to note the effect of indefinite repetition (*) with these commands:

R* /move the file pointer to the end of the current line.

L* /move the file pointer to the start of the current line.

E* /erase from the file pointer to the end of the current line.

E-* /erase from the start of the current line to the file pointer.

10. Breaking and joining lines

The following commands are used to break a line into two parts, and join two lines together.

B Break the current line into two parts at the file pointer. That is, insert a newline character immediately to the left of the file pointer, so that the second part becomes the new current line. B never fails. Breaking a line at its beginning (that is, when the file pointer is at the start of the line) has the effect of inserting an empty line immediately before it (which may also be achieved by means of G, then replying carriage return to the G prompt).

J Join the current line and the next line by removing the newline character at the end of the current line. This command fails if the current line already exceeds 120 characters in length. If the command succeeds then the file pointer is left at the join. Otherwise the file pointer is not moved.

11. Monitoring commands

There are three special commands that are used to change the amount of 'echoing' that ECCE performs. These have no effect on the source text. Special commands must appear as the only command in the command line.

&M Monitor normally. This is the default when ECCE is first invoked. The current line of the source text is typed at the console after the execution of each command line unless two conditions hold true. Firstly the current line has just been typed out in response to the immediately preceding command line, and secondly the current command line did not cause the file pointer to be moved across a line boundary at any stage in its execution. The current line is not re-echoed if the last command in the latest command line was a Print command.

&F Full monitoring is turned on. The current line is typed out after the execution of each command line unless the last command executed was a print (P) command. Thus the result of all command lines is displayed at the console.

&Q Quiet mode. Turn off all monitoring. The current line is typed out only if explicitly requested by means of the print (P) command, or if the command line fails.

12. The repetition command

A command line consisting solely of a number or * causes the previous command line to be executed the specified number of times. For this purpose repetition commands do not count as command lines, so that it is the most recent non-repetition-command command line that is repeated. Zero (0) or * causes the previous command line to be executed until failure. For example:

```
>M                          /move to the next line
>3                          /move 3 more times (if possible)
>2                          /move twice more (not 6 times)
>m-2                        /move back two lines
>4                          /now back 8 more (not 4)
>*                          /all the way to the start of the file
>P3                         /print the first 3 lines of the file
The cat
sat on
the mat.  Clever cat.
>                           /ready for next command
```

Note that this command sequence was executed in quiet mode (see above: monitoring commands) and the '>' is ECCE's command prompt. Text in the right hand half of the line after (and including) '/' is explicative and is not typed at the console.

13. Context specification - D, F, T, and U revisited

So far D, F, T, and U have been presented with their default contexts only. However, the number of lines to be searched in a text location command may be specified explicitly by inserting a number between the command letter and the text. This makes it possible to limit the scope of Find (by default the rest of the file), and extend the scope of Delete, Traverse and Uncover (by default the rest of the current line).

For example:

```
D50/DELETE/          /search 50 lines at most
F10/HELP/            /search 10 lines at most
T3/END/              /search 3 lines
U*/./                /search rest of file
```

The effect of specifying a context is best explained with reference to a specific command, for example, D50/DOG/. This operates in the following way.

(1) D1/DOG/ /search current line for DOG
(2) If found then Delete DOG and terminate the command successfully.
(3) Has the D command been executed 50 times?
(4) If so then command fails.
(5) Otherwide Move the next line and GO TO (1).

If ths command succeeds the effect is to delete the first occurrence of DOG within the next 50lines (current line + next 49 lines). If it fails the effect is M49 (Note, M49, not M50).

The other examples above have the following effects respectively. Find the first occurrence of HELP within the next 10 lines (or M9 if HELP doesn't occur within the next 10 linens), Traverse the first occurrence of END within the next 3 lines (or M2 if END does not occur within the next 3 lines), and more dangerously, Uncover all text up to the next '.' (or kill the rest of the file if '.' does not occur before the end of the file).

Note that this final command U*/./ is very dangerous, as all lines of text passed are killed.

These commands may also be repeated, for example.

```
F15/HELP/2
```

This command seeks the string HELP within the rest of the current line and the next fourteen lines, then, if successful, ignores the occurrence just found and seeks HELP again within the rest of the new current line and the fourteen lines following it. The command will fail if any repetition of F15/HELP/ fails, in which case the effect is either M14 if the first search fails, or M (between 14 and 28) if the first succeeds and the second fails. If the command succeeds the file pointer is left immediately before the second occurrence of HELP.

14. Programmed Command

Already the reader will have noticed that several commands may be put on one command line to create a comand 'program'. Also command lines may be repeated a specified number of times, or until failure. This section describes how more general 'programs' can be written.

( ) Bracketing. A string of commands bracketed together is treated as one command for purposes of failure and repetition. For example:

```
(M I/     /)*
```

inserts four blanks at the start of each line of the rest of the file.

[? Optional execution. Any command failure condition is ignored, for example:

```
D/PHONE/?
```

deletes the word PHONE from the current line if it occurs on the current line. Otherwise it does nothing.] not yet implemented

\ Inverted failure (success) condition. A command followed by \ instead of a repetition number has its failure condition inverted, i.e. it succeeds if and only if it fails! For example:

```
(MV/%R/\)0
```

This programmed command moves the file pointer to the start of the next line then repeats if V/%R/ fails. That is, the move is repeated until a line beginning with %R is found or until end of file is reached. This is one of the rare cases where SNUFF is more powerful, 0/%R/ is equivalent to the above example.

, (comma) Alternative command sequences. Sequences of commands separated by commas form alternatives. If the first alternative fails the next is tried, and so on. If an alternative succeeds then the following alternatives are ignored. This allows a generalised IF-THEN-ELSE construct to be programmed. For example

```
D/CAT/,D/DOG/,M
```

This command either deletes CAT or DOG on the current line, or moves the file pointer to the start of the next line (try to Delete CAT. If unsuccessful THEN try to Delete DOG. If still unsuccessful THEN try to Move to next line). Note too that command lines may be broken immediately after a comma and thus span two or more physical lines.

For example

```
>D/CAT/,
D/DOG/,M
```

Note that one must be very careful when using commands with alternatives. For example

MD/CAT/,MD/DOG/

would not have anything like the effect of the previous example. In this case if D/CAT/ fails we move to the next line and try D/DOG/. If DOG and CAT occurred on consecutive lines and the command were started on a line containing CAT (next line contains DOG) then no occurrences of CAT or DOG would be deleted.

By judicious use of () , ? \ and repetition, quite complex editing sequences can be programmed. The examples below are fairly difficult and the reader is invited to unravel then before looking at the answers.

```
(D/CAT/ I/CHAT/ L* , D/DOG/ I/ CHIEN/ L* , M)*

((T/X'/((RV/'/\)4,LV/'/S/'D/L,)4 E-2 I/16_/ D/'/))*M)*
```

The first 'program' is fairly straightforward and translates the words CAT and DOG to their french equivalents (but note that catalogue gets translated to chatalogue!). The second example changes hexadecimal numbers in the format X'n', X'nn', X'nnn', and X'nnnn' to 16_nnnn. That is it puts in the leading zeros to pad the field width to four. Note also the following useful commands.

```
(RM)0                       /find the next empty line
(RO(LO/ /)OM)O              /remove trailing spaces from all lines
(R*(LD//)*M*
((I/bbbb/M)60B6)*
((I/    /M)6DB6)O           /right shift all lines by four spaces
                            /and separate pages with 6 empty lines
```

This final command is useful for paginating a file before printing it on a line-printer, but note the assumption of 66 lines per page.

15. Macros

Three macro commands can be defined by the user, namely X, Y, and Z. When the effect of a macro is exactly as if the macro body had been typed instead of the macro invocation. For example

```
>&X=(RM)O              /note %X=<macro-body>
                       /not  X=<macro-body>
```

This defines X to be the next-empty-line command (see above). The effect of X in a command line is then exactly the same as (RM)*. Macro body length is restricted to 63 characters; in addition, both macro bodies and command lines may contain no more than 40 command units where each comma, bracket, \, number, and simple command counts as one unit.

It should be noted that when a macro is invoked the macro body exactly replaces the macro name. For example if X were to be defined as MK then

```
X4
                  is exactly equivalent to
MK4
                  and not to
(MK)4
                  as might be imagined.  But note that
(X)4
                  is equivalent to
(MK)4
```

Note that a macro can be defined as several command lines separated by semicolons. For example

```
&X=%s;f/&ROUTINE/nf/%END/TEXT 2
```

16. Additional control commands

&N - switches on line number printing.

&NOFF - switches off line number printing.

&I = [import file,][[FROM=]<line no>,][[TO=]<lineno>]*

- inserts the request import file at the current cursor position.

&S switch between normal and secondary input.

&R Rewind to top of file, updating the output file.*

&R,S Rewind secondary file.

&T <tab character>[,<integer position>]

&T Displays tab setting

&P[-/+] turns input prompt off/on

* Not yet implemented on MVS

APPENDIX

Alphabetical command summary

The following section is an alphabetical summary of ECCE commands. It is split into three sections, the special commands which begin with a % character and must occur one to a command line, the programmed command qualifiers, and the simple commands. The paragraph references are to the earlier paragraphs of this document.

Special commands

&C	Close the editing session (paragraphs 2, 4).
&F	Full monitoring (paragraph 11).
&M	Monitor normally (default). (paragraph 11).
&Q	Quiet mode. No monitoring (paragraph 11).
&X=	Macro definition (paragraph 15).
&Y=	Macro definition (paragraph 15).
&Z=	Macro definition (paragraph 15).
&A	Abort the edit session (paragraph 4)
&R	Rewind to top of file, updating the output file.
&S	Switch between normal and secondary input.
&I	= [import file,][[FROM=]<line no>,][[TO=]<lineno>]
&N	Switches on line number printing.
&T	Displays tab setting.

Programmed command qualifiers

( )	Bracket a group of commands for purposes of repetition and failure (paragraph 14).
[?	Optional execution of command (paragraph 14).[not yet implemented
\	Invert the failure condition (succeed if an only if the command fails) (paragraph 14).
,	Alternative execution (IF-THEN-ELSE....) (paragraph 14).

Simple commands

B	Break. Insert a newline at the current position (paragraph 10).
D/TEXT/	Delete the first occurrence of TEXT (paragraphs 6, 13).
E	Erase the next character (paragraph 9).
E-	Erase the previous character (paragraph 9).
F/TEXT/	Find the first occurrence of TEXT (paragraphs 4, 6, 13).
G	Get a line of input (paragraph 4).
I/TEXT/	Insert TEXT at the current position (paragraph 6).
J	Join the current line to the next line (paragraph 10).
K	Kill the current line (paragraph 4).
L	Left shift the file pointer (paragraph 5).
M	Move the file pointer to the next line (paragraph 4).
M-	Move the file pointer to the previous line (paragraph 4).
P	Print the current line (paragraph 4).
R	Right shift the file pointer (paragraph 9).
S/STR/	Substitute STR for the TEXT just Found, Uncovered, or Verified (paragraph 6).
T/TEXT/	Traverse TEXT. Move the file pointer past the next occurrence of TEXT (paragraphs 6, 13).
U/TEXT/	Uncover TEXT. Remove all characters between the file pointer and the next occurrence of TEXT (paragraphs 6, 13).
V/TEXT/	Verify that TEXT occurs immediately to the right of the file pointer (paragraph 6).
C	Move cursor to specified column in line (paragraph 4)
H	Move cursor to specified column in line. Moving test to the right (paragraph 4).
X	Macro invocation (paragraph 15).
Y	Macro invocation (paragraph 15).
Z	Macro invocation (paragraph 15).