University of Edinburgh

Department of Computer Science

ECCE

the

Edinburgh Compatible Context Editor

by

L.D. Smith

Internal Report CSR-24-78

James Clerk Maxwell Building
The King's Buildings
Mayfield Road
Edinburgh
EH9 3JZ

May, 1978

ECCE

the

Edinburgh Compatible Context Editor

ECCE 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 people within both the department of Computer Science and the Edinburgh Regional Computer Centre have contributed (to) implementations of ECCE.

L. D. Smith. 18/05/78

Contents

Introduction

ECCE - the Edinburgh Compatible Context Editor - is a program for creating, updating, and extending text files. Any line-structured file can be manipulated by ECCE (but there could be problems if too many of the characters in the file are non-printing!). Examples of line-structured files include IMP and FORTRAN source files, and the source text for documents such as this.

ECCE is implemented on most of the computers in both the Computer Science Department and the Regional Computing Centre, and also on the Science Research Council's DEC system-10.

1. The source file

Conceptually all editing operations are performed on a file. Actually a source file is edited to an output file, leaving the source unchanged, although often (or indeed usually) the output file will have the same name as the source file and will overwrite or replace the source file when ECCE terminates successfully. This, of course, has the same apparent effect as directly editing the source file.

2. Commands

After invoking ECCE, commands are issued from a console, one to a line or many to a line, and in either upper or lower case. Spaces have no significance, except within text strings, and command lines are terminated either by a newline character or by a semi-colon (;). 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 and close all input and output files 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 before the end of file marker. 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.
```

The end of file marker is identified at the console by the editor's typing **END**.
For example

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

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 <u>or</u> 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 allow 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. That is, until the command fails or a large, implementation dependent, number of repetitions have been performed. Typically this large number is set at 5000.

%C	Close the editing session. The input and output files are closed and control returned to the invoking system. %C must be the only command in the command line.
M	Move the file pointer to (just before) the start of the next line. This fails if the file pointer is already at the end of file marker (that is, after the last line of the file).
M-	Move the file pointer back to (just before) the start of the previous line. This fails if the file pointer is already at, or in, the first line of the file. It may also fail because of implementation restrictions on how far back the file pointer may be moved (on systems with only sequential files the limit is determined by the available buffer space).
K	Kill the current line of text. That is, remove all characters in the current line (including the end of line character). 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.
G	Get a line 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 gotten 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.
F/TEXT/	Find TEXT. Search the rest of the file, starting at the file pointer, for the first occurrence of the string TEXT. / is a string delimiter. Any character having no other significance to the editor can be used for example ' or " or . or $ (or / itself). 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).

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 simple 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 file marker (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 file marker 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.

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 a '/' character.

System prompts and invocation of ECCE are, of course, system dependent (see section 3: Invoking ECCE - system dependencies) and an arbitrary example is given.

```
   xE N/POEM                    /invoke the editor (system dependent)
   Edit                         /to make a new file
   >g0                          /get some lines of input
   :Discretion
   :Discretion is the better part of Valerie
   :(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 puddings 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**
   >g0                          /and continue the input ...
   : ...
```

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 many 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 an end of line is encountered before the text is found. Find fails only at the end of the file. In subsequent searches for the text string, by any text location command, the occurrence just found is ignored. The commands Find, Uncover, and Verify are text location commands. The Delete command breaks this rule and will delete the occurrence just found, however, it is not a text location command since the specified text is removed from the file.

In all the following examples the '/' character is used as a string delimiter. Any character with no other significance to the editor could be used, for example ' or " 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/ or 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, a V, or a 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 text location command (cannot use Substitute after it). Traverse is extremely 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 within the current line after the file pointer. 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 pointer. The file pointer is left immediately after the inserted TEXT. Insert may fail in some implementations of ECCE owing to restrictions on the line length and/or the command line length. 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 is 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 occurrs immediately to the right of the file pointer. Fail otherwise. If successful, S/STR/ can be used 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 combinations of commands are often found to be useful

```
F/TEXT1/ S/TEXT2/
D/TEXT1/ I/TEXT2/
T/word/ I/ending/
U/./                 /delete rest of sentence
U/./S/,/             /delete rest of sentence
                     /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 comment and monitoring conventions, and the command and input prompts, are exactly the same as in the previous example.

```
   xE N/POEM                  /invoke the editor (system dependent)
   Edit                       /to make a new file
   >g0                        /get some lines of input
   :Discretion
   :Discretion is the better part of Valerie
   :(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 puddings made of rice
   :she takes carbohydrates like God takes advice
   :a surfeit of ambition
   :is her particylar vice
   :Valerie fondles lovers
   :like a mousetrap fonles mice
   ::                         /end the input for now

   >m-0                       /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.B.  D is not a text looation command
                              /so 'y' just found is deleted
   >f/nle/ s/ndle/
 * like a mousetrap fondle^s mice
   >%c                        /end the edit

   x                          /next command prompt from system
                              / (system dependent)
```

8. Command failure

A command can fail in two ways. Firstly it may be syntactically incorrect, in which case the command line is ignored and an error report produced, and 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 successful 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
          >mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
           SIZE?               /command too long
          >g/abc/
          G /?                 /wrong syntax entirely
```
but note
```
          >w;m                 /W (rest of command line) is ignored
                               /but M (next command line) is executed
```

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* 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 at a finer level and move the file pointer past one character, or delete one character, of the source text.

R	Right shift the file pointer one character position. This fails if the file pointer is already at the end of a line.
L	Left shift the file pointer past one character position. This fails if the file pointer is already at the beginning of a 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 a 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 a line.

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 command).
J	Join the current line and the next line together by removing the newline character at the end of the current line. This command fails if the new line length would exceed the maximum allowed by the implementation. 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)
>0                    /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 that '>' 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. That is one line, with the exception of F which operates on the rest of the file (see paragraph 6.: Text location and manipulation commands). In fact the number of lines to be searched by these commands can be specified explicitly. 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) Otherwise Move to next line and GO TO (1)
```

If this command succeeds the effect is to delete the first occurrence of DOG within the next 50 lines (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 lines), 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, or M28. If the command succeeds the file pointer is left immediately before the second occurrence of HELP.

14. Programmed commands

Already the reader will have noticed that several commands may be put on one command line to create a command '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/    /)0
```

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.

\ Inverted failure (success) condition. A command followed by \ has its failure condition inverted, and 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 the repetition limit is exceeded (or until end of file is reached).

, (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 <u>no</u> 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 them <u>before</u> looking at the answers.

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

   ((T/X'/((RV/'/\)4,(LV/'/S/'0/L,)4 E-2 I/16_/ D/'/))0M)0
```

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 zeroes to pad the field width to four. Note also the following <u>useful</u> commands

```
   (RM)0                        /find the next empty line
   (R0(LD/ /)0M)0               /remove trailing spaces from all lines
   ((I/    /M)60B6)0            /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.

Standard extensions

There are two widely implemented standard extensions to ECCE, macros and secondary input. The implementation details of secondary input vary, and two variants will be described.

15. Macros

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

```
          >%X=(RM)0               /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)0. It should be noted that it is the length of the macro body that counts towards the total command line length. Both macro body length and command line length are restricted, typically to 64 characters and 40 command units (each comma, bracket, \, number, and simple command counts as one unit). These limits are, however, very implementation dependent.

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.
```

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

```
          %X=f1/TEXT1/;f1/TEXT2/
```

16. Secondary input

Secondary input is a feature that allows parts of a second input text to be merged with the source text. The secondary input text is completely unchanged by any editing operations that might be performed upon it.

To enter secondary input mode the command %S is issued, and from this point ECCE prompts for commands with '>>' rather than '>'. To leave secondary input mode another %S command is issued and ECCE then prompts for commands with '>' again. A %C command issued in secondary input mode has the same effect as in primary input mode, and ends the editing session in the same way. A %S command must be the only command in the command line.

While in secondary input mode ECCE maintains a second file pointer to the secondary input text, and any editing commands issued in secondary input mode cause the second file pointer to be moved. The main file pointer remains at the point it was at when the %S command was issued. If secondary input is re-entered then the secondary file pointer is where it was when secondary input mode was last left. Initially the secondary file pointer is at the beginning of the secondary input file.

Most systems implementing secondary input (but not PDP9/PDP15) do so by means of two more commands, Note, and Abstract. These are issued in secondary input mode. The effect of Note is to note the current position of the (secondary) file pointer. This overrides any previously issued Note commands. The secondary file pointer can then be moved using (in principle) almost any of ECCE's commands. Some implementations prohibit commands that would normally insert or remove text (for example I, S, D, U, K, G) and only allow the file pointer to be moved. In addition, the EMAS implementation prohibits the use of character manipulation commands. The effect of an Abstract command is then to transfer all text from just after the last Noted position to just before the current (secondary) file pointer into the output text in a position just before the (main) file pointer. The Abstract command may be repeated with the obvious effect. Note that when an Abstract command is issued, the secondary file pointer must be after the last Noted position, otherwise it fails.

On the PDP9/PDP15 systems secondary input is managed in a different manner. The details of leaving and entering secondary input mode are identical, but there are no Note or Abstract commands. Instead, all text passed by the secondary file pointer will be included in the output file, and text that is not to be included must be Killed, Erased, Uncovered, or Deleted before the file pointer is moved past it. Only text occurring before the secondary file pointer is inserted into the output file. It is not possible to move the secondary file pointer backwards.

Examples

```
>%s                  /enter secondary input mode
>>f/read tag/        /move to position
%ROUTINE ^READ TAG(%INTEGER NAME)
>>mm-                /to get to start of line!
>>n                  /note the position
>>(v/%end/\m)0m      /move past end of routine

>>a                  /abstract the routine
>>%s                 /and back to main source
>                    /usual command prompt
```

In the PDP9/PDP15 implementation this would be

```
          >%s                 /enter secondary input mode
          >>(f1/read tag/\k)0 /kill until read tag
          %ROUTINE ^READ TAG(%INTEGER NAME)
          >>(v/%end/\m)0      /move to end of routine
          %END
          >>m                 /past end of routine

          >>%s                /back to main source
          >                   /main prompt
```

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. Commands peculiar to specific implementations of ECCE are described in the next paragraph.

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).
%S	Secondary input mode (paragraph 16).
%X=	Macro definition (paragraph 15).
%Y=	Macro definition (paragraph 15).
%Z=	Macro definition (paragraph 15).

Programmed command qualifiers

( )	Bracket a group of commands for purposes of repetition and failure (paragraph 14).
?	Optional execution of command (paragraph 14).
\	Invert the failure condition (succeed if and only if the command fails) (paragraph 14).
,	Alternative execution (IF-THEN-ELSE...) (paragraph 14).

Simple commands

A	Abstract all text between the last Noted position and the file pointer (paragraph 16).
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 9).
M	Move the file pointer to the next line (paragraph 4).
M-	Move the file pointer to the previous line (paragraph 4).
N	Note the current position (paragraph 16).
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).
X	Macro invocation (paragraph 15).
Y	Macro invocation (paragraph 15).
Z	Macro invocation (paragraph 15).

Invoking ECCE - system dependencies

Interdata

ECCE is invoked on the Computer Science Department's Interdata systems by a command of the form

```
E source/output
```

Secondary input and macros are not available in this implementation. If the source filename is omitted, or the null filename N specified, then a new output file is created. If the output filename is the same as the source filename, or it is omitted, then the output file overwrites the source file when the edit is closed. If the output filename is the null filename N then all output is thrown away (this is a convenient way to examine a file with no danger of altering it). For example

```
E FRED/NEWFRED              /FRED is unchanged
E MYPROG                    /MYPROG updated at end of edit
E /NEWPROG                  /create a new file NEWPROG
E LIST/N                    /examine LIST
```

PDP9 or PDP15

ECCE is invoked on the Computer Science Department's PDP9/15 systems by a command of the form

```
E source,secondary-input/output
```

Secondary-input may be omitted if not required. If the source filename is omitted, or the null filename N specified, then a new output file is created. If the output filename is the same as the source filename, or it is omitted, then the output file overwrites the source file when the edit is closed. If the output filename is the null filename N then all output is thrown away (this is a convenient way to examine a file with no danger of altering it).
For example

```
E MYPROG/NEWPROG            /MYPROG unchanged
E PROGRAM                   /PROGRAM updated at end of edit
E PROG1,PROG2/DT3 MERGED    /merge parts of PROG2 with PROG1
E /DT5 NEW FILE             /create NEW FILE on DT5
E DT2 LIST/N                /examine LIST on DT2
```

EMAS

Before using ECCE on EMAS it is neccessary to append CSDEPT.EDLIB. It is then invoked by a command of the form

```
E(source,secondary-input/output)
```

If the source filename is omitted, or the null filename .NULL specified, then a new output file is created. If the output filename is the same as the source filename, or it is omitted, then the output file overwrites the source file when the edit is closed. If the output filename is the null filename .NULL then all output is thrown away (this is a convenient way to examine a file with no danger of altering it). For example

```
E(.NULL/NEWFILE)            /create NEWFILE
E(FRED/FRED2)               /FRED unchanged
E(FRED)                     /FRED updated at end of edit
E(PROG1,PROG2/MERGED)       /merge parts of PROG2 with PROG1
E(LOOK/.NULL)               /examine LOOK
```

ICL 2980

ECCE is invoked on the ICL 2980 by calling the macro ECCE with parameters INPUT=, OUTPUT=, SECIN=, CONTROL=, LISTING=, and RESPONSE=. These specify the input, output, secondary input, control, and listing streams. SECIN may be omitted if not required, and OUTPUT defaults to the next generation of the input file (generation -3 is deleted to prevent the accumulation of old versions). CONTROL defaults to the job-stream if not specified and contains the commands to control ECCE. LISTING is the name of a file to receive messages and error reports from ECCE. By default a workfile is used which is automatically listed. RESPONSE specifies the name of an integer SCL variable which will receive the result code at the end of the edit. The default is RESULT. The result code takes one of two possible values, namely 0 if there are no errors, and 1 if the edit is abandoned because of an edit command failure.

Users wishing to use ECCE on the ICL 2980 should consult the ERCC program advisory service for more detailed information.

Commands peculiar to the ICL 2980 implementation of ECCE

M#	Move to absolute line number 'n'. This command fails if there is no line 'n'.
K#n	Kill all lines from the current line up to, but not including, the line with absolute line number 'n'. This command fails if the file pointer is already past line 'n' or if line 'n' cannot be found. In the latter case all lines from the current position up to some line with line number greater than 'n' are deleted.

O/TEXT/	Observe TEXT. Move the file pointer to the first occurrence of TEXT, outputting all lines between the current file pointer position and TEXT to the console. This command fails if TEXT does not occur before the end of the file.
%A	Abandon the edit. The editing session is abandoned with the input file unchanged. If a Rewind command has been issued then the input file reverts to the state it was in when last rewound.
%P	Print line numbers. When a line is printed it is preceded by its line number enclosed in parentheses. If the line has been created (not from input file) the line number is printed as four blanks. This is the default mode of operation for ECCE on the 2980.
%N	No line numbers to be printed.
%D	Display last line output. Due to restrictions on memory size only a portion of a file can be held in memory. As the file pointer is advanced through the file earlier portions of the file have to be output to backing store and cannot be re-edited. The %D command notifies the user of the last line output to indicate how much of the file is available for re-editing.
%R	Rewind. The remainder of the input is copied to the output file and both are closed. The newly generated file then becomes the input file with the file pointer at the start of it. Absolute line numbers now refer to positions in this new file.
%O=n	'n' is an unsigned integer. The absolute record numbers are adjusted so that the current record has record number 'n'. The command is ignored if the current record does not possess an absolute line number.

DEC system-10

ECCE is invoked on the Science Research Council's DEC system-10 by a command of the form

```
ECCE output=source,secondary-input
```

Secondary-input may be omitted if not required, in which case a second copy of the source is made available as secondary-input. This facility effectively allows blocks of text to be moved around in a file, by copying the text from the copy of the source to the new position in the output, then deleting the same block from the old position in the output. Note that in secondary input mode, only the five commands M, M-, F, T, and P, and the secondary input mode commands N and A, are valid.

If the source filename is omitted then a new output file is created. If the source filename is the same as the output filename,

or it is omitted, then the output file overwrites the source file when the edit is closed. If the output filename is omitted then all output is thrown away (this is a convenient way to examine a file with no danger of altering it).
For example

```
   ECCE NEW.TXT=                 /create NEW.TXT
   ECCE FRED.TXT=SAM.TXT         /SAM.TXT unchanged
   ECCE MERGED.TXT=OLD.TXT,NEW.TXT
                                 /merge parts of NEW with OLD
   ECCE PROG.TXT                 /PROG.TXT overwritten by new version
   ECCE =LOOK.LST                /examine LOOK.TXT
   ECCE                          /use last used ECCE parameters
```

There are several features of ECCE that are peculiar to the DEC system-10 implementation. These are described below.

Whenever a file being modified by the ECCE is closed then a backup copy with extension .BAK is made.

The switches /F/L/M/N/P/Q/U and /E are allowed after a file specification to set the special modes %F, %L, %M, ... (see below, or command summary) and /E specifies that input and output files are to be encrypted and decrypted using the standard DEC-10 encryption routines. ECCE asks for a password (up to 30 characters) for each file. A null password (carriage return) specifies that no encryption is to be done on that file.

At the start of execution, ECCE looks for the file ECCE.CMD in the user's own file area, and, if it is found, starts reading commands from that file. Thus special commands, macros, or even whole editing sessions, can be defined. When a command file has been found ECCE types out the message "Reading from command file".

The execution of a command line can be interrupted from the console by typing control-C twice followed by REENTER. This forces an immediate failure condition for the command line. Thus an inadvertent K* could be aborted (if noticed soon enough) after killing only a few lines rather than the whole file. The file pointer is left at the line in which it was when execution was interrupted.

Further details of the use of ECCE on the DEC system-10 can be obtained from the DEC-10 user support group.

Commands peculiar to the DEC system-10 implementation of ECCE

%A	Abort. Close all files, but do not rename the originals, leaving the altered file with the extension .TEM.
%G	Go. Close all files and execute the last COMPIL class command.
%T	Top. Reset the file pointer to the top of the file and save all the editing so far in a file with extension .TMP.

%W	Write. Save all the editing so far in a file with extension .TMP but do not move the file pointer.
%N	Normal case mode. Characters in text strings or input using the G command are taken in the case in which they are given (default).
%L	Lower case mode. All characters in text strings, or input using the G command, are converted to lower case.
%U	Upper case mode. All characters in text strings, or input using the G command, are converted to upper case.
%P	Print mode. All non-printing control characters in the file being edited are typed at the console as an up-arrow (or caret) character followed by the appropriate letter, for example tab=^I, cr=^M, lf=^J. When in this mode the file pointer is typed as an underscore character _.

PDP11, DEIMOS

ECCE is invoked under DEIMOS by a command of the form

```
E source/output
```

Secondary input and macros are not available in this implementation of ECCE. If the source filename is omitted, or the null filename N specified, then a new output file is created. If the output filename is the same as the source filename, or it is omitted, then the output file overwrites the source file when the edit is closed. If the output filename is the null filename N then all output is thrown away (this is a convenient way to examine a file with no danger of altering it). For example

```
E FILE/NEWFILE          /FILE is unchanged
E MYFILE                /MYFILE updated at end of edit
E /NEWFILE              /create a new file NEWFILE
E LIST/N                /examine LIST
```

PDP11, DOS

ECCE is invoked under DOS by a command of the form

```
R ECCE
```

The program then prompts "*"; type the filenames in the form

```
output=source
```

If the source filename is omitted then a new output file is created. If the source filename is the same as the output filename, or it is omitted, then the output file overwrites the source file when the edit is closed. If the output filename is omitted then all output is thrown away (this is a convenient way to examine a file with no danger of altering it).

For example
```
   * NEW=                    /create file NEW
   * NEW=OLD                 /edit from OLD to NEW
   * FILE                    /FILE updated at end of edit
   *.=LIST                   /look at LIST
```

Other systems

Several implementations of ECCE have been written in IMP (for example, EMAS, PDP9/15) and so ECCE is easily implementable on any system which supports IMP.