E C C E - A Text Editor

E C C E - A Text Editor

CONTENTS

CONTENTS

APPENDICES

Authors/Acknowledgements:

Software: Murray Langton, ULCC, adapted from a Pascal program supplied by the University of Nottingham, with assembler routines for input and output which use software supplied by the University of Cambridge.

Document: Murray Langton, based on documents and machine-readable text from the Universities of Oxford, Glasgow, and Nottingham, with constructive criticism from Dave Tilley and Gyan Mathur of ULCC.

Original publication date: February 1987

1 INTRODUCTION

INTRODUCTION

ECCE is a command-driven text editor which is available at many U.K. universities on a variety of computers. This document describes ECCE as implemented by ULCC on IBM and IBM-compatible computers under the MVS operating system.

ECCE has a relatively small number of basic editing commands so that learning to use ECCE is fairly straightforward. ECCE has its own internal help system so that a list of available commands, or details of any particular command, can be obtained at any time while editing. The basic editing commands may be combined, repeated, and made conditional so that more sophisticated editing operations, such as global text changes in specified contexts, can readily be performed.

As well as being used as an editor to create or alter a file, ECCE may also be

used to scan a file without changing anything. The input/output routines used by ECCE can handle most MVS file formats, so that ECCE may also be used to copy files and to convert files from one format to another (including conversion between fixed and variable length records, insertion or removal of carriage control characters, and changes in block size).

ECCE stands for Edinburgh Compatible Context Editor. It was originally devised by Mr. Hamish Dewar at the University of Edinburgh. No new editing commands have been added for the implementation by ULCC, but some commands have been extended and some extra options have been introduced to cope with the MVS environment (see Appendices B and G). Due to operating system restrictions one special command (%R) has not been implemented (see Appendix G).

This document is a complete manual for ECCE; most users will only need to read selected portions. Both an informal tutorial (Section 3) and a more detailed reference guide (Section 4) are included, together with reference summaries in the Appendices.

Users new to ECCE are advised to glance through Section 2, to work through
new
Section 3 at a terminal in order to gain some experience with using ECCE, and then to read Section 2 more thoroughly, with occasional references to Section 4 for more detailed explanations of commands of interest.

Users with some experience of ECCE may find it sufficient to read Section 2,
some experience
Appendices A, B, G, either C or D, and consult parts of Chapter 4 as required.

Special software, known as DECCE, is available for some intelligent terminals which allows ECCE to be used as if it were a full-screen editor (see Appendix F).

While ECCE is normally used from an interactive terminal, it can also usefully be used in a batch job, though greater care is needed to avoid errors (see Appendix E).

2 CONCEPTS and GENERAL INFORMATION

CONCEPTS and GENERAL INFORMATION

Throughout this document editing commands are shown in CAPITAL LETTERS
CAPITAL LETTERS
to make them more visible; in fact, ECCE will accept editing commands in any mixture of upper-case and lower-case letters.

Throughout this document double diamond brackets '(( ))' have been used to indicate extra information which can be ignored by most users but which may be of interest to those curious users who wish to know the full story.

2.1 The Editing Process

The Editing Process

As an editing session progresses, ECCE makes an internal copy of the file being edited. All changes are made to this internal copy. No changes are made permanent until a user explicitly tells ECCE that editing is complete; only when this happens is the internal copy transferred back to disc storage. Alternatively, a user may tell ECCE that editing is to be abandoned, in which case the internal copy is discarded, leaving the original file unchanged.

Only a finite amount of space is available to hold the internal copy. Users who wish to edit files more than 4,000 lines long should read Section 4.6. Note however that ECCE does not store trailing spaces within its internal copy, not
so that files of more than 4,000 lines may well fit within the available space if they contain short lines.

((The internal copy only contains text up to the last line actually referenced by a user during an editing session. When editing is complete, the internal copy is first transferred to the output file, and then any untouched text in the input file is transferred directly to the output file. This process is invisible to the user but is noticeably more efficient when changes are only made near the start of a large file.))

2.2 Context Pointer

Context Pointer

ECCE maintains a (notional) pointer which refers to the current position in the pointer current position
file being edited. The line within which the pointer is positioned is known as the current line.
current line.

This pointer is positioned between characters rather than on a character. At between
the beginning of a line the pointer is positioned just before the first character on that line. At the end of a line the pointer is positioned just after the last character on that line (ignoring trailing spaces).

Editing commands are available to move the pointer either within the current line or to some other line. Except for very large files (see Section 4.6) the pointer can be moved at random to any position in the file. Most actual editing is done just before or just after the pointer.

2.3 Original Line Numbers

Original Line Numbers

As lines are read from the (primary) input file, ECCE assigns a number to each line for reference purposes during that editing session. Lines are numbered 1, 2, 3, etc. which is helpful when editing in conjunction with a line-numbered listing. ((The maximum line number is 65536.))

These numbers are known as original line numbers, and remain attached to a line
original
throughout an editing session. Any lines created or inserted by the user are treated as unnumbered. The pointer can be moved directly to the start of any numbered line.

2.4 How Lines are Displayed

How Lines are Displayed

When the current line is displayed to the user the position of the pointer is shown by a commercial at symbol '@', except when the pointer is at the start of the line.

By default, ECCE displays the current line whenever a new line becomes current, so that a user can keep track of position in the file. Alternatively, a user may request that the current line be displayed after every command line or that the current line never be displayed except on explicit request.

When a line is displayed it is normally preceded by its original line number (enclosed in round brackets). For unnumbered lines the brackets enclose spaces to maintain column alignment. A user may suppress the display of line numbers.

The end of a file is indicated by an imaginary line '**END**' which has a line number but which does NOT form part of the file.

Lines may be up to 256 characters long. Lines which are longer than the width of a terminal can still be edited normally, but will occupy more than one line when displayed. Users may specify that only part of each line should be displayed.

((When displaying lines, ECCE uses a vertical bar '|' to represent actual tab characters, and uses a question mark to represent all other control characters. Note that the control characters are not changed to a question mark or vertical
not
bar within the text being edited. There may be other unprintable characters which ECCE cannot detect, since terminals and printers vary widely with regard to the characters which they can handle.))

2.5 Files Used

Files Used

ECCE normally uses four files, but may use between three and eight files, depending on which special facilities are being used. The use made of the various files is described below.

Primary Input File ((DD name=SRCE))
Primary Input File

This file holds the text to be edited or scanned. It may be null if a file is being created. It may in some cases be replaced by the output file if editing concludes successfully.

Command File ⟨⟨DD name=EDIT⟩⟩
Command File
This is the file from which edit commands are read. For interactive use, it is normally the user's terminal. For batch use, it is normally included in-line with the job control statements.

Report File ⟨⟨DD name=LIST⟩⟩
Report File
This file is used for displaying lines being edited and for issuing error or warning messages. For interactive use, it is normally the user's terminal. For batch use, it is normally the printer.

Output File ⟨⟨DD name=DEST⟩⟩
Output File
This file holds the result of editing. It is not produced if ECCE is merely being used to scan a file.

Secondary Input File ⟨⟨DD name=SECSRCE⟩⟩
Secondary Input File
This file is optional; it provides an additional source of text. All or parts of the secondary input file may be merged with the primary input file using standard editing commands. The primary and secondary input files may in fact be the same file if it is desired to re-arrange text within that file. The %I command also allows file to be merged.

Initial Command File ⟨⟨DD name=INIT⟩⟩
Initial Command File
This file is optional; it provides an additional source of edit commands, which are processed before the standard command file. This allows some user-defined commands to be processed before commands are obtained from the user, so that each user may alter the normal ECCE defaults to suit themselves. This file could also be used to hold macro definitions.

⟨⟨There are two other files which ECCE may use; the DD names are HELP and P@DUMP. The 'HELP' file contains text from which ECCE extracts help information requested by users. The 'P@DUMP' file is used in the event of a serious internal error to produce a formatted dump; if this ever appears, then the dump should be referred to ULCC Advisory for analysis.⟩⟩

2.6 Command Prompts

Command Prompts

At a terminal, ECCE always provides a visible prompt whenever it is waiting for a user to type something. Three different prompts are used so as to indicate what ECCE is expecting the user to type.

Prompt	User action

> Enter an editing command which applies to the primary input file.
>

>> Enter an editing command which applies to the secondary input file
>>
(the command %S switches between primary and secondary). For more
%S
details, see Sections 3.5 and 4.4.4.

: Enter a line of text to be inserted into the file before the current
:
line (to terminate text entry and return to a command prompt, just type a colon ':' as the only character on a line).

2.7 How to Interrupt ECCE

How to Interrupt ECCE

ECCE may be interrupted by a standard MVS attention interrupt. This facility
attention
is really only useful if a repeated command is to be forcibly stopped.

The exact way in which a user generates an attention depends on the terminal and on how it is connected to the mainframe computer. ((For teletype-like terminals connected via a PAD the normal attention sequence is CTRL/P B while for 3270 full-screen terminals the PA1 key (ALT/DUP) is used.))

In response to an attention interrupt, ECCE will display: *** ATTN (ECCE):
*** ATTN (ECCE):
and wait for a user response. A user may type N to ignore the attention, or
N
may press the RETURN key to cause ECCE to return to command mode. The first
RETURN
command issued should normally be P to find out where the pointer is, or %A
P %A
to abandon editing.

2.8 How to Terminate ECCE

How to Terminate ECCE

There are three ways in which a user can tell ECCE that editing is finished. In response to a command prompt a user should enter one of the following special commands:

%C Conclude editing successfully. A new output file is created
%C
containing the edited file.

%A Abandon editing. The input file remains unchanged and no new output
%A
file is created.

%K %K Conclude editing successfully. A new output file is created containing the edited file, up to but excluding the current line. The current line and all subsequent lines are discarded.

2.9 Editing Commands and Repetition

Editing Commands and Repetition

Basic commands consist of a single letter which may be followed by additional Basic information. For example, M is a basic command which means move to the beginning of the next line, P is a basic command which means print the current line (display it at the user's terminal), and M- is a basic command which means move to the beginning of the previous line.

The actions performed by the basic commands are usually fairly simple. To perform more complicated actions it may be necessary to combine several basic commands. Any number of basic commands may be placed on a command line (optionally separated by spaces).

When several commands are placed on a single command line, ECCE checks the entire line for syntax errors, such as mismatched delimiters or brackets, before obeying any commands from that line. Note that ECCE converts all commands internally to upper-case so that an error message may refer to say 'M' when the user had typed 'm'.

Any basic command may be followed by a number to specify a repetition count, so that M2 means move forward two lines. An asterisk '*' may be used instead of a number to specify indefinite repetition, so that M-* means move back as far as possible (usually to the first line of the internal copy). ((Indefinite repetition really means 'up to 5,000 times' to prevent infinite loops.))

A compound command consists of any number of basic commands enclosed in round brackets '( )'. For example (PM3)* means print out every third line from a file, starting with the current line. Compound commands may be repeated and nested.

Special commands consist of a percentage sign '%' followed by a letter and possible additional information. Usually only one special command may appear on each command line. Simple examples of special commands are %C and %A as detailed in Section 2.7. As well as terminating editing, special commands may

be used to set and display various options, to define simple macros, and to insert text from another file.

2.10 String Delimiters

String Delimiters

In some of the edit commands specified later, character strings must be specified. For example, F/rubbish/ is an editing command to find the character string 'rubbish'. To avoid ambiguity, these character strings must be delimited (preceded and followed) by a delimiter character. Possible delimiter characters include the following:

F/rubbish/

/ . : @ ' "

The delimiter chosen must not, of course, appear within the character string being delimited. For the examples in this document, a slash (solidus, divide symbol) '/' is used as a delimiter. It is suggested that either '/' or '.' should normally be used as a delimiter since these are convenient to type.

((In fact, any character other than those treated specially by ECCE may be employed as a delimiter. ECCE ignores spaces and uses the special characters % ( ) * , - ; ? \ as well as letters and digits.))

2.11 Command Success and Failure

Command Success and Failure

In some cases ECCE may not be able to obey a command given by the user. For example, the pointer can not be moved left when it is already at the beginning of a line. In such circumstances the command is said to fail and ECCE will report the failure to the user. The precise conditions under which each command may fail are detailed in Section 4.

fail

Note that failure is not necessarily a bad thing; indefinite command repetition with '*' stops when the command fails (without any failure report), and failures can be used within compound commands to terminate loops or select context-dependent action (see Section 4.3).

3 TUTORIAL ON USING ECCE

TUTORIAL ON USING ECCE

3.1 About This Tutorial

About This Tutorial

This tutorial is intended to provide a gentle introduction to editing using ECCE. Note that this tutorial concentrates on the simple use of simple commands. Only about half of the basic ECCE editing commands are used, and only a few of the special commands are used. Refer to Appendix A for a full list of available commands. and then to Section 4 for more details of any particular command of interest.

This tutorial may be used in conjunction with either Phoenix 3 or TSO. Note that ECCE as described in this document is not available under Phoenix 1. ((TSO at ULCC is neither recommended nor supported, and so is used at the user's own risk; in practice a standard TSO service is available for those users already familiar with TSO.))

Before attempting this tutorial you must be registered as a user (or have the legitimate use of an account number). You should also know how to connect your terminal to ULCC and how to log on to either Phoenix 3 or TSO. Remember that the computer does NOT act on what you type until you press the key labelled 'RETURN' (or, on some terminals, 'ENTER').

When Phoenix 3 or TSO is ready for you to type a command it displays a 'ready' prompt; for Phoenix 3 the prompt is 'PHX ready' while for TSO it is 'READY'. ((These are the normal 'ready' prompts; some experienced users may have defined an alternative prompt.))

3.2 Creating a File

Creating a File

To create a file using ECCE you should type one of the following commands in response to the 'ready' prompt. Under Phoenix 3, '$' represents an null or empty file. ((The effect of this command will be to create a partitioned data set called 'DEMO' with a member called 'MARY' into which the file being created will be placed.))

Phoenix 3: ECCE $ TO .DEMO:MARY

TSO: ECCE DEMO(MARY)

ECCE will respond with a header line which contains a version number, current date and time, and instructions on how to use the ECCE help system. This is followed by an ECCE command prompt ')'. ((Under TSO you will also get a message saying that a new file is being created.))

```
ECCE/ULCC/1.0    Wed 17 Sep 1986 11:55    For help type %H
)
```

Before proceeding further you may like to type %H to see what happens. You will get a list of 'help' topics, mainly single letter command names, plus terse instructions on how to ask for help on a particular topic. (For example, %H

for help on topic 'INTRO' type %H INTRO).
%H INTRO

Following the information provided by %H will be the single line
%H
(0001)**END** This indicates that the current line is the (imaginary) end of file; since this is line number one, the file itself is empty. The current line is printed by default after each command line you type unless it has already been printed.

Now type G* which tells ECCE to get an indefinite number of lines from the
G*
terminal and insert them in the file before the current line. ECCE will prompt you with a colon ':' and you should enter the following lines. Please copy any spelling mistakes since later sections of this tutorial will use ECCE to correct them. The fifth line that you enter is just a colon ':' which tells ECCE to stop entering lines and prepare to accept another command.

```
:Mary had a litle lamb
:Its fleace was white as sno
:And everywhere that Mary went
:The lamb was shore to go.
::
)
```

In response to the command prompt ')' now type %C which is the special
%C
command to conclude editing and actually write the file onto disc storage. When this has been done you will then receive the 'ready' prompt again.

3.3 Simple Editing

Simple Editing

The next stage is to correct the mistakes in the file. In response to the 'ready' prompt type one of the following commands. ((The alert reader may notice that the TSO command is exactly the same as that used to create a file: the difference is that this time the file already exists.))

Phoenix 3: ECCE .DEMO:MARY

TSO: ECCE DEMO(MARY)

You will then get the ECCE header line followed by the ECCE command prompt ')'. Type the special command %F to tell ECCE to provide full monitoring, which
%F
means that the current line is to be printed after every command line has been obeyed. This makes it easy to check your edits.

Now type P to print the current line, which will be the first line of the
P

P
file. This line contains one error ('litle' instead of 'little'). To correct this error type F/litle/S/little/ which means find the character string

F/litle/S/little/
'litle' and substitute 'little' for it.

Note that the current position of the pointer is shown by a commercial at symbol '@'. If you find this symbol obtrusive then type the command L* which means move the pointer left as far as possible. The position of the pointer is not displayed when it is at the start of a line.

Type M to move to the next line which should contain two errors ('fleace'

M
instead of 'fleece', and 'sno' instead of 'snow').

To correct the first error you could type F/fleace/S/fleece/ but it would

F/fleace/S/fleece/
be sufficient in this case to type F/a/S/e/ since 'F/a/' will uniquely locate

F/a/S/e/
the error. You could obviously correct the second error in a similar fashion by typing F/sno/S/snow/ but you could instead type R*I/w/ which means

F/sno/S/snow/ R*I/w/
move right as far as possible (to just after the last visible character on the line) and then insert the character 'w'.

You could use two successive M commands to move from the current line (line

M
2) to line 4, or you could type P3 which will print out 3 lines starting with

P3
the current line, or you could type the command M#4 which means move directly

M#4
to line number 4. Line 4 contains one error ('shore' instead of 'sure'). Again you could correct this error by typing F/shore/S/sure/ or you could

F/shore/S/sure/
instead type D/ho/I/u/ which means find and delete the characters 'ho' and then insert the character 'u'.

D/ho/I/u/

To check the file type M-*P* which means move back as far as possible and

M-*P*
then print out as much as possible. To conclude the editing type %C so that

%C
ECCE can transfer the edited file from its internal copy back to permanent disc storage.

Important: At this stage Phoenix 3 has merely created a new current data set.
Important:
An additional Phoenix 3 command is required to overwrite the original file, as follows:

Phoenix 3: FILE TO .DEMO:MARY

3.4 Editing to Another File

Editing to Another File

In this section we will edit the file to make a different verse and put this verse in a separate disc file. In response to the 'ready' prompt type one of the following commands. When using Phoenix 3 the '/SHR' suffix is needed when members of the same partitioned data set are being used as both input and output files.

Phoenix 3: ECCE .DEMO:MARY TO .DEMO:MARY1/SHR

TSO: ECCE DEMO(MARY) TO(DEMO(MARY1))

Again in response to the ECCE command prompt ')' you should type the command %F to switch full monitoring on. The changes required are in lines 2 and 4 of %F
the original file.

Type M#2F/white/S/black/F/snow/S/soot/ which will move directly to line 2 and
M#2F/white/S/black/F/snow/S/soot/
then change 'white' to 'black' and 'snow' to 'soot'.

The changes to line 4 are more extensive so that it is probably easiest to remove line 4 and retype it completely. Type M#4KG to move to line 4, kill
M#4KG
(delete) it, and then get a single line from the terminal, which you should type as follows:

```
:Its sooty foot it put.
)
```

Again you may check the entire file by typing M-*P* before typing %C to
M-*P* %C
conclude editing. No special 'FILE' command is needed with Phoenix 3 in this case since an explicit 'TO' data set was provided.

3.5 Joining Two Files Together

Joining Two Files Together

In this section of the tutorial the two files created will be joined together ((concatenated)) to make a more complete poem. In response to the 'ready' prompt type one of the following commands.

```
Phoenix 3:  ECCE .DEMO:MARY AUXIN .DEMO:MARY1

TSO:        ECCE DEMO(MARY) SEC(DEMO(MARY1))
```

In response to the ECCE command prompt ')' type either P* or M* to get to the end of the file DEMO(MARY). Now type the special command %S to switch to the secondary input file DEMO(MARY1). This switch is shown by a change in the command prompt from ')' to '))'.

Any command you now type applies to the secondary input file and any command which moves through that file cause the line(s) moved past to be inserted in front of the current line of the primary input file. Hence, to copy the whole file just type P* or M* again. Now type %S to switch back to the primary input file (and prompt ')').

On typing M-*P* you will get the entire poem displayed, and can then see that there is no gap between the verses. To insert a blank line, use the commands M#4MB to move to line 4, then move to the following unnumbered line, and then split that line before the first character i.e. insert a blank line.

You can now type M-*P* to display the complete poem before typing %C to conclude the editing. Again, when using Phoenix 3, the following command is required to transfer the result of editing from the current data set back to a specific disc file:

```
Phoenix 3:  FILE TO .DEMO:MARY
```

3.6 Systematic Global Editing

Systematic Global Editing

If you are feeling slightly overwhelmed, then this section of the tutorial may be omitted, or deferred to a subsequent editing session. It is intended to give you some idea of how to use simple loops to perform systematic editing throughout a file. In response to the 'ready' prompt type one of the following commands.

```
Phoenix 3:  ECCE .DEMO:MARY TO .DEMO:MARY2/SHR

TSO:        ECCE DEMO(MARY) TO(DEMO(MARY2))
```

In response to the ECCE command prompt type (F/lamb/S/sheep/)* and then type M-*P* to look at the poem; all occurrences of 'lamb' have been changed to

M-*P*
'sheep'!

Any number of simple commands may be enclosed in round brackets to form a compound command and then this compound command can be repeated as if it were a compound command
simple command. ((Compound commands can be nested to any reasonable depth.))

As another example of systematic editing, type the command M-*(I/... /M)*M-*P*
M-*(I/.../M)*M-*P*
which will move back to the start of the poem, insert three dots and a space at the beginning of each line, and then display the result.
This next example is a simple demonstration of context-dependent editing. The intention is to examine each line of the file in turn, and to replace the three dots by three exclamation marks for all lines that contain the word 'Mary'. Type the command M-*(F1/Mary/L*E3I/!!!/M,M)*M-*P*
M-*(F1/Mary/L*E3I/!!!/M,M)*M-*P*

The F1/Mary/ command just attempts to find 'Mary' within at most 1 line (i.e. F1/Mary/
on the current line) and fails if 'Mary' is not found. When a command fails within a compound command, control transfers to the next comma in that compound command, if it exists, and to the end of the compound command otherwise. Note particularly that each alternate sequence in the above compound command ends with M in order to ensure that progress is made through the file; it is very M
easy to set up a loop which does nothing to the same line indefinitely.

This last example just displays all lines which start with an exclamation mark, using the command M-*(V/!/PM,M)
M-*(V/!/PM,M)

The V command just checks if the specified character string immediately V
follows the pointer and fails if it does not. You might like to experiment with a few more compound commands before concluding the edit by typing %C as %C
usual.

3.7 Deleting a File

Deleting a File

As a result of the above tutorial you now have a partitioned data set called 'DEMO' with three members called 'MARY', 'MARY1' and 'MARY2'. In response to a 'ready' prompt you can delete an individual member by typing one of the following commands.

Phoenix 3: DELMEM .DEMO:MARY1

TSO: DELETE DEMO(MARY1)

You can delete the entire partitioned data set and all its members by typing

one of the following commands.

Phoenix 3: DELETE .DEMO

TSO: DELETE DEMO
4 REFERENCE GUIDE

REFERENCE GUIDE

4.1 Notation and General Information

Notation and General Information

Most sections in this chapter contain a group of related commands. Within each section commands are arranged in alphabetical order. Where a command has several variations the simplest form of each command is shown first. To reduce repetition, obvious failure conditions are not always given for all variations.

To simplify the command descriptions it is assumed that files contain less than 5,000 lines and that the whole file will fit within the space available for ECCE's internal copy. For larger files refer to Section 4.6.

In the command examples /xx/ represents a text string 'xx' delimited by '/'. For full details of permissible string delimiters refer to Section 2.9. 'm' and 'n' represent optional numbers; where meaningful '*' may be used instead of a number to indicate 'indefinite' repetition ((up to 5,000 times)). ((ECCE will actually accept '0' instead of '*', which may be more convenient to type in some cases, but is certainly not as clear!))

((In most sensible cases, ECCE treats semicolons ';' within a command line as equivalent to end-of-line; however, a semi-colon is treated as an ordinary character within delimited strings, inserted lines and the special commands %T %V %W %X %Y %Z. Use of a semicolon may allow several special commands to be placed on one line and it can also reduce the number of interactions required. For example, in response to the command prompt, type KG;any text
KG;any text
to replace the current line by 'any text'.))

4.2 Basic Commands

Basic Commands

Basic commands consist of a single letter, which may in some cases be followed by additional information. Any number of basic commands may be placed on one line (optionally separated by spaces).

Where sensible, most basic commands may be repeated by following them with a number. For example, F/xx/ will find the next occurrence of 'xx', while
F/xx/
F/xx/3 will find the third occurrence of 'xx'.

F/xx/3

A number (or asterisk) can be typed as a command line. This causes the previous command line to be repeated a specified (or indefinite) number of times.

Some commands may move the pointer one or more lines; if such a command fails then the pointer is normally left at the start of the line on which the command actually failed.

4.2.1 To Move the Pointer

Cn — Move the pointer just before the 'n'th character on the current line. If there are less than 'n' characters on the current line then trailing spaces will be inserted up to column 'n'.
FAILS if the maximum line length would be exceeded or if n = 0.

F/xx/ — Find the next occurrence of 'xx' and place the pointer immediately before it.
FAILS if 'xx' does not occur before the end of file is reached. In this case the pointer is placed at the beginning of the imaginary **END** line.

F/xx/n — Find 'n'th occurrence of 'xx'.

Fm/xx/ — Find next occurrence of 'xx', searching at most 'm' lines.

Fm/xx/n — Find 'n'th occurrence of 'xx', searching at most 'm' lines for each occurrence. 'n' defaults to 1 and 'm' defaults to *.

L — Move the pointer left one character position.
FAILS if pointer is already at start of line.

Ln — Move left 'n' character positions.

L* — Move left to beginning of current line (never fails).

M — Move forward to the start of the next line.
FAILS if pointer is already at the end of the file, i.e. at the imaginary **END** line.

Mn — Move 'n' lines forwards through the file.

M* — Move to the end of the file (never fails).

M- — Move back to the start of the previous line.
FAILS if the current line is the very first line.

M-n — Move back 'n' lines.

M-* — Move back to the start of the file (never fails).

M#n — Move directly to the start of the line numbered 'n'.
FAILS if no line is found with the appropriate line number, in which case the pointer is left at the start of the line with the next higher number.

O/xx/ — (observe text) This has the same effect on the pointer as the F command described above, except that all lines from the current line to the line containing the text are displayed.
FAILS if 'xx' does not occur before the end of file is reached.

Om/xx/n — Observe 'n'th occurrence of 'xx', searching at most 'm' lines for each occurrence. 'n' defaults to 1 and 'm' defaults to *.

P Print the current line (display it at the terminal).

Pn Print 'n' lines starting with the current line. Note that the single command P does not move the pointer, but that a repeated command such as P3 is actually treated as equivalent to PMPMP, P3 PMPMP so that the pointer finishes up at the start of the last line printed.

FAILS if an attempt is made to print past the end of file.

P* Print lines until the end of file is reached (never fails).

P#n Print lines starting at the current line until a line with original line number 'n' is reached (or passed).

R Move the pointer right one character position.

FAILS if pointer is already at end of line.

Rn Move right 'n' character positions.

R* Move right to end of line (never fails).

T/xx/ (traverse text) Move the pointer just after the next occurrence of 'xx' on the current line.

FAILS if 'xx' is not found before the end of the current line.

T/xx/n Move the pointer just after the 'n'th occurrence of 'xx' on the current line.

Tm/xx/ Move the pointer just after the next occurrence of 'xx', searching at most 'm' lines. 'm' may be '*' to search the rest of the file.

Tm/xx/n Move the pointer just after the 'n'th occurrence of 'xx', searching at most 'm' lines for each occurrence. 'n' and 'm' both default to 1.

4.2.2 To Insert or Alter Text

G Get a line from the terminal (or command file) and insert it before the current line. At a terminal the required line is prompted for with a colon ':'.

FAILS if a line containing just a colon is typed.

Gn Get 'n' lines and insert them before the current line.

FAILS if a line containing just a colon is typed.

G* Get an indefinite number of lines and insert them before the current line. Insertion stops when a line containing just a colon is typed.

Hn (hop) Insert spaces before the current pointer position until the pointer is just before column 'n'.

FAILS if the pointer is already past column 'n'.

I/xx/ Insert 'xx' before the pointer.

FAILS if the maximum line length would be exceeded.

I/xx/n Insert 'n' copies of 'xx' before the pointer.

S/xx/ Substitute 'xx' for the characters just found by an F, O, U or V command. The pointer is left just after the text which has been

substituted.
FAILS if the last command which moved the pointer was not F, O or U or if V has not just been used. Also fails if the maximum line length would be exceeded.

4.2.3 To Delete Text

D/xx/ Delete the next occurrence of 'xx' on the current line. The pointer is moved to the position which the deleted text occupied.
FAILS if 'xx' is not found on the current line.

D/xx/n Delete the next 'n' occurrences of 'xx' on the current line.

Dm/xx/ Delete the next occurrence of 'xx', searching at most 'm' lines. 'm' may be * to search the rest of the file. If this fails then the pointer is at the start of the last line searched.

Dm/xx/n Delete the next 'n' occurrences of 'xx', searching at most 'm' lines for each occurrence. 'n' and 'm' both default to 1.

E Erase character immediately to the right of the pointer.
FAILS if pointer is at end of current line.

En Erase 'n' characters to the right of the pointer.

E* Erase all characters after the pointer on the current line.

E- Erase character immediately to the left of the pointer.
FAILS if pointer is at start of current line.

E-n Erase 'n' characters to the left of the pointer.

E-* Erase all characters before the pointer on the current line.

K (kill line) Delete the current line. The pointer is moved to the start of the line after that deleted.
FAILS if at end of file.

Kn Delete 'n' lines starting with the current line.

K* Delete the current line and ALL following lines.

K#n Delete up to, but excluding, the line whose original line number is 'n'.
FAILS if the pointer is already past the specified line or the specified line is not found. If the specified line can not be found, then lines are deleted until a line is found whose original line number exceeds 'n'.

U/xx/ (uncover text) Delete all characters from the pointer up to but excluding 'xx' on the current line.
FAILS if 'xx' is not found on the current line.

U/xx/n Delete all characters up to but excluding the 'n'th occurrence of 'xx' on the current line.
FAILS if 'xx' is not found on the current line.

Um/xx/ Delete all characters up to but excluding the next occurrence of 'xx', searching at most 'm' lines. 'm' may be * to search the rest of the file.

FAILS if 'xx' is not found within 'm' lines, but lines will still be deleted up to but excluding the one at which the command fails.

Um/xx/n Delete all characters up to but excluding the 'n'th occurrence of 'xx', searching at most 'm' lines for each occurrence. 'n' and 'm' both default to 1.

4.2.4 To Break and Join Lines

B (break) The current line is split in two at the position of the pointer; the second part becomes the current line with the pointer at its start. If the pointer is at the beginning of a line then the effect is to create a blank line before the current line.

Bn Repeat B 'n' times i.e. split the current line as for B and then insert 'n-1' blank lines between the two portions. If the pointer is at the beginning of a line then the effect is to insert 'n' blank lines before the current line.

J (join) The next line is appended to the current line. The pointer is left at the join.

FAILS if the maximum line length would be exceeded.

Jn Join the next 'n' lines to the current line.

4.3 Compound and Alternate Commands

Compound and Alternate Commands

The following basic command is frequently used in compound commands as a means of testing for character strings of interest.

V/xx/ Verify that 'xx' is immediately to the right of the pointer. This command does not alter the text or move the pointer.

FAILS if 'xx' does not follow the pointer.

Any number of basic commands may be enclosed in round brackets to form a compound command. Compound commands may be repeated and nested. A compound compound
command fails when any command within it fails, except in the case of alternate commands described below. If a complex compound command is to be used several times in an edit, then use can be made of the macro facility described in Section 4.5.

Examples of compound commands:

(commands)n Execute the commands 'n' times, with individual commands being executed in order from left to right. If any individual command fails than no further commands are executed and the compound command fails.

(commands)* Execute the commands indefinitely as above, until an individual command fails, and then terminate the repetition without failure.

(MR)* Move to the start of the next blank line.

(F/sun/S/moon/)* Replace 'sun' by 'moon' throughout the remainder of the file.

(F/sun/S/moon/P)* As above, and display all lines altered.

(F/sun/S/moon/L*P)* As above, and display all lines altered, without displaying the position of the pointer.

A comma ',' may be used to separate alternate command groups, which may alternate themselves be basic or compound. If the first command group fails, control passes to the next alternative without a failure being reported. Should that alternative fail, control passes to the next alternative, and so on, until either

(a) a command group is completed successfully, in which case all further alternative command groups are ignored, or
(b) no further alternate command groups are available, in which case the entire command fails.

It is important to ensure that, in the case of indefinite repetition, some progress is made through the file regardless of which alternate command group is obeyed. Otherwise the effect may be to do nothing to the same line indefinitely, which can be a bit tedious. Refer to Section 2.7 for details of how to interrupt ECCE.

Examples of alternate commands:

M-*(V/C/K,M)* Delete all lines in a file which start with 'C'.

M-*(V/C/M,K)* Delete all lines in a file which do not start with 'C'.

M-*(F1/begin/L*PM,F1/end/L*PM,M)* Display all lines in a file which contain either 'begin' or 'end'.

A back-slash '\' may be used to invert command failures. When a command is invert followed by a back-slash instead of a repetition number, the failure criterion is inverted. Thus, a successful execution fails while a normally failing one does not.

A question mark '?' may be used to cancel command failures. When a command is cancel followed by a question mark instead of a repetition number, any failure of that command is ignored.

Examples of failure inversion and cancelling:

(MV/T/\)* Find the next line starting with 'T'.

E? Erase 1 character if not at end of line.

(E8)? Erase 8 characters or until end of line.

M-*((R72E*)?M)* Truncate all lines to at most 72 characters.

((V/UP/I/S/)?M)* Insert 'S' before all lines starting with 'UP'.

M-*(F/cat/(E-)?)* Throughout a file, delete the character just before 'cat', except at the beginning of a line.

(F1/work/\K)* Delete all lines up to but excluding the next line containing 'work'.

M-*((R81L*P)?M)* Display all lines containing more than 80 characters.

M-*((V/hurry/\,(R61L*P)?)M)* Display all lines starting with 'hurry' which contain more than 60 characters.

4.4 Special Commands

Special Commands

Special commands consist of a percentage sign '%' followed by a letter and possible additional information. Usually only one special command may appear on each command line ((though in some cases semi-colons may be used, as described in Section 4.1)).

4.4.1 To Terminate Editing

%A Abandon editing. The input file remains unchanged and no new output file is created.

%C Conclude editing successfully. The internal copy and the remainder of the primary input file (if any) are copied to the output file.

%K Text up to but excluding the current line is transferred from the internal copy to the output file. Any other text in the internal copy or in the primary input file is discarded and not transferred. ((This is equivalent to K*;%C but is more efficient.))
K*;%C

4.4.2 To Control Display of Lines.

Note that the commands in this section only affect the way in which lines are displayed; They do not affect the actual editing process in any way.

%F (full monitoring) The current line is to be displayed after execution of each command line, unless the last command executed was a print command. This is particularly useful when editing since it provides immediate feedback by displaying the effect of any changes

to the current line.

%J=m Display lines from column 'm' onwards.
%J=m,n Display only columns 'm' to 'n' inclusive.
%J=,n Display up to and including column 'n'.
%J= Revert to default; display complete lines.
%J? Display current display margin settings.

%M (default monitoring) The current line is displayed only when a new line becomes current. This is sufficient to allow a user to keep track of position in the file.

%N Suppress display of line numbers.

%P Enable display of line numbers (default).

%Q (quiet) Suppress all automatic monitoring. Lines are only displayed when a user enters a P command.

%T=char (see next section) Among other things, this command specifies the visible character to be used to display any genuine tab characters in the file. The default is a vertical bar '|'.

4.4.3 Miscellaneous

%B ON Make blanks (spaces) significant when matching character strings. This is the default.
%B OFF Ignore blanks when matching character strings.
%B Inverts the current status of the significance of blanks.

%D Display the absolute line number of the line most recently written to the output file. This value will normally be zero, except when the file being edited is too big to fit within the space available for ECCE's internal copy (for large files see Section 4.6). This indicates how far back in the file M-* will move.

%H Display a list of topics on which help is available.
%H topic Display information on 'topic'.

%O=n Redefine line numbers. The current line is made to have line number 'n' and all other numbered lines are adjusted to match. Note that line numbers are internal to ECCE. If this would cause any line to have a number less than 1, then that line is treated as unnumbered.

%T=char,n1,n2,...,n Define pseudo-tab character and tab stops. Up to 50 tab stops may be defined.
%T=n1,n2,...,n Define tab stops, leave pseudo-tab character unchanged.
%T=char Define pseudo-tab character, leave tab stops unchanged.
%T= Default setting: no pseudo-tab character, tab stops at columns 7, 10, 13, 16, 19, 22, 25, 28, 31, and 41
%T? Display current tab setting.

This special command makes it easy to input text which must be

aligned at particular columns. The effect of this command applies to any lines of text that are read in by subsequent G commands. Any occurrence of the pseudo-tab character is replaced by one or more spaces so that the following character will be placed at the column defined by the next tab stop.

For example, the effect of:

```
)%T=@,7,10,15,25
)G2
:is@@a bell@tolling @?
:@@harken@ye
```

would be to insert text aligned as follows:

```
  (123456789012345678901234 5 column numbers for reference)
   is    a bell  tolling   ?
            Harken         ye
```

Another effect of the %T command is to alter the way in which genuine tab characters within a file are displayed. By default, any genuine tab characters are displayed as a vertical bar '¦'. But if a pseudo-tab character has been defined, then this is also used when displaying genuine tab characters.

%V=c Define a visible character 'c' as a visible blank (space). Any occurrence of the visible blank character in the command file is replaced by an actual blank (space) character before ECCE processes the line. This facility makes it easier to type a specific number of blanks. For example, after using the command %V=? the command I/??????x??????/ would insert 6 blanks, an 'x', and then 6 more blanks.

%V= Remove any definition of a visible blank character.

%V? Display current visible blank character, if any.

%W=c Define a 'wild character' for use in matching character strings. This wild character will match any character. For example, after the command %W=? the command F/a = ??/ would find any of 'a = bc', 'a = de', 'a = ?a', etc.

%W= Remove any definition of a wild character.

%W? Display current wild character, if any.

4.4.4 For Inserting Text From Other Files

%I=filename Insert a file before the current line.

%I=filename,FROM=..,TO=.. Insert part of a file.

This special command inserts text from the specified file. A series of such commands enables several files to be concatenated. The full file name must be specified (e.g. ABCD123.FRED or ABCD123.PDS(MEM1) or ABCD123.TEXT:MEM2). If no filename is specified then the primary input file is used.

The pointer is first moved to the start of the current line; lines read from the specified file are inserted before the current line.

The parameters FROM= and TO= can be used to select part of the file. The value for FROM/TO may be either a line number or text enclosed by a string delimiter. FROM defaults to the beginning of the file and TO defaults to the end of the file. A text string specifies the next line which contains the supplied text string. For example, to insert from line 10 of the file 'test' to the next line containing 'end of', the command used could be %I=test,FROM=10,TO="end of"

%R,S Rewind the secondary input file. See also %S. The next reference to the secondary input file will start at the beginning of that file. The secondary input file is never changed. This command allows the secondary input file to be reordered while it is being merged with the primary input file.

%S Switch to or from the secondary input file. See also %R,S. This command enables the text from two files to be concatenated or selectively merged. The primary and secondary input files may in fact be the same file; this allows a file to be rearranged or duplicated. Before being switched either way, the pointer is first moved to the start of the current line, and that position is 'remembered' in case of a subsequent reverse switch.

The secondary input file is never actually changed, but editing commands may be used to select which portions of the secondary input file are actually inserted. Any non-destructive forward motion through the secondary input file causes the characters and/or lines moved over to be inserted before the current line of the primary input file. Any destructive forward motion through the secondary input file causes that text to be skipped rather than actually deleted.

The pointer may only move forward through the secondary input file, and the lines are treated as unnumbered. Thus M#, P#, K#, M-, L, and E- cannot be applied to the secondary input file.

4.5 Defining and Using Macros

Defining and Using Macros

```
%X=text   Define macro X.
%Y=text   Define macro Y.
%Z=text   Define macro Z.

%X?       Display current definition of X.
%Y?       Display current definition of Y.
%Z?       Display current definition of Z.
```

These special commands are used to define and display the text of three macros called X, Y and Z. The characters between the equals sign '=' and the end of the line form the text of the macro.

After the macro in question has been defined, when a corresponding X, Y or Z character is encountered in a suitable position in a command line, it is effectively replaced by the defined text of the appropriate macro. Macros can include calls on other macros ((and indeed on themselves, but macro calls nested more than 3 deep are ignored without comment)). The maximum number of characters in the text of a macro is 63. Macros can be re-defined at any time.

For example after %Y=F/ABC/PM-3 the command line R3YP2 is equivalent to
%Y=F/ABC/PM-3 R3YP2
R3F/ABC/PM-3P2. As another example, after %X=F/ABC/I/D/ the command line X5
R3F/ABC/PM-3P2 %X=F/ABC/I/D/ X5
is equivalent to F/ABC/I/D/5. Note that only the I command is repeated 5
F/ABC/I/D/5
times. To repeat all the commands in X, it is necessary to either use (X)5
(X)5

or to define %X=(.....).
%X=(.....)

Note that the text for X, Y and Z need not be complete commands. X, Y and Z are not commands, but signals to replace X, Y or Z by the appropriate section of text. Substitution only occurs when the X, Y or Z to be substituted for can be immediately recognised as not belonging to the text within a text string. Hence %Y=OW/ will not be substituted within F/BYM2.
%Y=OW/ F/BYM2

4.6 Editing Large Files

Editing Large Files

This section only applies to users who are editing large files. The actual definition of 'large' depends on the length of lines in the file, since ECCE does not store trailing spaces within its internal copy. As a rough guide, there is space for about 4,000 lines of 80 characters each, or about 7,500 lines of 40 characters each, or some roughly equivalent values. ((For each line ECCE requires an additional 6 characters to hold control information.))

For such large files, ECCE uses a technique known as 'windowing', whereby the space available for the internal copy is used as a window which moves down the file. If the space available for the internal copy should become full, then some text is automatically written to the output file in order to make room for more text from the input file. Thus the available space acts as a moving window on the file being edited.

Access is only allowed to that portion of the file which is within the window. In particular, M-* will only move back to the start of the window, rather than to start of file. Hence, when editing large files, it is important to remember to make amendments more or less sequentially rather than in random

order. The windowing process will then remain invisible.

Users should also note that indefinite repetition with '*' really means 'repeat up to 5,000 times', so explicit repetition counts may be needed when dealing with files containing more than 5,000 lines.

APPENDIX A Summary of ECCE Commands

In the summary below 'n' represents a number and /xx/ represents a text string 'xx' delimited by '/'.

Command	Section	Brief description
B	4.2.4	Split current line into two at pointer position.
Cn	4.2.1	Move pointer to precede column 'n' of current line.
D/xx/	4.2.3	Delete next occurrence of 'xx' on current line.
E	4.2.3	Erase character immediately to right of pointer.
E-	4.2.3	Erase character immediately to left of pointer.
F/xx/	4.2.1	Place pointer just before next occurrence of 'xx'.
G	4.2.2	Obtain line from user, insert before current line.
Hn	4.2.2	Insert spaces until pointer is at column 'n'.
I/xx/	4.2.2	Insert text 'xx' before pointer.
J	4.2.4	Append next line to end of current line.
K	4.2.3	Delete current line.
K#n	4.2.3	Delete lines up to but excluding line number 'n'.
L	4.2.1	Move pointer left one character position.
M	4.2.1	Move pointer to start of next line.
M-	4.2.1	Move pointer to start of previous line.
M#n	4.2.1	Move pointer to start of line number 'n'.
O/xx/	4.2.1	As for F, and display all lines passed.
P	4.2.1	Display current line, '@' indicates pointer.
P#n	4.2.1	Display from current line up to line number 'n'.
R	4.2.1	Move pointer right one character position.
S/xx/	4.2.2	Text 'xx' replaces text found by F, O, U or V.
T/xx/	4.2.1	Place pointer just after next 'xx' on current line.
U/xx/	4.2.3	Delete text on current line from pointer to 'xx'.
V/xx/	4.3	Verify that text 'xx' follows pointer.
X	4.5	Insert macro text defined by %X.
Y	4.5	Insert macro text defined by %Y.
Z	4.5	Insert macro text defined by %Z.
%B ON	4.4.3	Blanks significant when matching character strings.
%B OFF	4.4.3	Blanks not significant when matching character strings.
%D	4.4.3	Display number of last line written to output file.
%F	4.4.2	Full monitoring: display current line except after P.
%H	4.4.3	Display list of HELP topics.
%H xx	4.4.3	Display helpful information on topic 'xx'.
%I=..	4.4.4	Insert all or part of another file.
%J=n,m	4.4.2	Display columns 'n' to 'm' of lines.
%M	4.4.2	Partial monitoring: display current line once only.
%N	4.4.2	Do not display line numbers.
%O=n	4.4.3	Renumber lines, with current line as line number 'n'.
%P	4.4.2	Display line numbers.
%Q	4.4.2	Quiet. No monitoring: use P to display current line.

APPENDIX B Options Available When Invoking ECCE

Some of the options described below may be found on many ECCE systems, though not necessarily in exactly this form. Other options are peculiar to this implementation of ECCE and are provided to help cope with the way in which files are stored under MVS. The way in which these options are specified varies slightly, depending on the environment in which ECCE is invoked (see Appendices C, D, E).

One default option is ASIS. If both the primary input file and the output file have carriage control characters, then the default is EDCC, otherwise the default is NOCC.

Option	Effect
ASIS	Accept all letters as they are, whether they come from a file or from the terminal. See CAPS below.
CAPS	Convert all letters to upper-case. See ASIS above. This applies to all text typed by the user and to all text obtained from the primary or secondary input file. This option is particularly useful when creating or editing a file which is required to contain only upper-case letters and where the keyboard being used does not possess a 'caps lock' key.
EDCC	Edit carriage-control characters. See NOCC below. Under MVS, a file which is intended for printing usually has an ANSI carriage control character in the first column of each line ('1'=new page, ' '=new line, '+'=overprint, etc.). Selecting EDCC makes these characters appear just like any other character for editing purposes. If the input file does not have carriage-control characters then spaces will be provided.
HUSH	Suppress information messages. ECCE normally announces itself with a header line giving its version number, date and time, and instructions on how to obtain help. Other information messages may also be produced in some circumstances. This option allows these messages to be suppressed, though any error messages will still appear. ((The HUSH option is really provided for those situations where ECCE is being called as a utility by some other program or command procedure.))
NOCC	Hide carriage-control characters. See EDCC above. If this option is selected then any carriage-control characters in the input file are discarded as the input file is read, and spaces are inserted, if required, as carriage-control characters in the output file as it is being written.

SCAN Just look at a file. In some cases, such as examining output or preparing to modify a program, it is convenient to examine a file via an editor without making any changes. For instance it is then easy to find all occurrences of a particular identifier. When SCAN is selected ECCE proceeds as usual, except that no output file is actually produced.

SHOW Echo edit commands. In some cases ECCE may be invoked at a terminal with some or all of the editing commands being taken from a file. The commands from the file are not normally displayed. The SHOW option causes the edit file to be displayed at the terminal with appropriate prompts inserted. This makes it easier to debug such a file.

APPENDIX C Using ECCE Via the Phoenix 3 Command Language

The ECCE command edits from one dataset to another. If no explicit destination is specified then the result of editing is a new current data set - this must be filed to a permanent file to keep the edited version.

Simple example to edit the file .DATA:A1
Simple example

```
ECCE .DATA:A1        - invoke ECCE
. . .
edit commands        - perform editing
. . .
%C                   - conclude editing (result in current dataset)
FILE TO .DATA:A1     - overwrite the original
```

Full syntax of ECCE command.
Full syntax

```
ECCE {{FROM} dsd} {TO dsd} {WITH dsd} {VER dsd} {INIT dsd}
     {AUXIN dsd} {OPT text} {DD text}

There are also keywords STORE, TIME, RCLEVEL and ERRLABEL.

Defaults:  ECCE FROM %C TO %O WITH *!$ VER * AUXIN $
           where  *!$  means  * if online, $ if in a batch job.
Thus:
   ECCE              edit from current dataset to new current dataset.
   ECCE .FILE        edit from  .FILE  to a new current dataset.
   ECCE .F1 TO .F2   edit from  .F1  to  .F2

Where:
   FROM   primary input file
   TO     output file
   WITH   command file
   VER    report file
   INIT   initial command file, obeyed before WITH.
   AUXIN  secondary input file (used by %S command)
   OPT    options as described in Appendix B (NOT editing commands)
```

```
      DD     set up other ddnames

      Note that if the SCAN option is selected then it is advisable to
      use the command    ECCE .FILE TO %N OPT SCAN
      otherwise a new empty current data set will be created.
                    empty

   ECCE return codes:
      16  Pascal runtime failure (should not occur).
      17  abandoned with %A command
      18  empty output file
      19  unable to open required files
APPENDIX D  Using ECCE Via a TSO Command
```

APPENDIX D Using ECCE Via a TSO Command

Simple examples (any data set names can be used):

```
   ECCE FRED              Edit and replace data set FRED
   ECCE FRED TO(JOE)      Edit data set FRED to data set JOE
   ECCE FRED SCAN         Use ECCE to scan but not alter data set FRED
```

Full command with all options shown:
(only 'dsn' is required, all other parameters are optional,
mutually exclusive parameters are separated by slash '/',
default values are shown in brackets)

```
   ECCE dsn TO() SEC() WITH() INIT()/OBEY()     name files to be used
        SCAN CAPS/ASIS HUSH EDCC/NOCC SHOW      various options
        DCB(VAR)/LIKE() SIZE(5) INC(5) DIR(5)   use if creating file
        ERRCODE(8) TRACE(0) VERSION(1)          (for expert use only!)
```

dsn — Specify name of primary input data set.
If 'dsn' does not exist, then it is assumed that 'dsn' is to be created (and DCB,LIKE,SIZE,INC,DIR may be used).
If 'dsn' does exist and if neither SCAN nor TO() is used, then a temporary data set will be created, which will replace 'dsn' if editing concludes successfully.

Various forms of data set name are acceptable –

	Convention	data set name	actual name used
(a)	TSO	FRED	UserId.FRED
(b)	TSO	'SYS1.HELP'	SYS1.HELP
(c)	ULCC	:SYS1.HELP	SYS1.HELP
(d)	Phoenix	.FRED	UserId.FRED
(e)	TSO	FRED(A)	UserId.FRED(A)
(f)	Phoenix	.FRED:A	UserId.FRED(A)
(g)	TSO/Phoenix	FRED:A	UserId.FRED(A)

TO(dsn2) — Specify name of output data set.
If 'dsn2' already exists, then it must be empty.
If 'dsn2' does not exist (the normal case), then it will be created (and DCB,LIKE,SIZE,INC,DIR may be used).

'dsn2' may be specified in the same way as 'dsn'.

If both 'dsn' and 'dsn2' are members of the same partitioned data set, then 'dsn2' may also be specified in one of the following forms:

(h) TO((member))
(i) TO(.(member))
(j) TO(.:member)

SEC(dsn3) Specify name of secondary input data set.
'dsn3' may be specified as for 'dsn2'

WITH(dsn4) Specify name of edit command file.
This replaces the terminal as the source of edit commands and is useful mainly when ECCE is being used either as a utility by other programs or when the same commands are to be applied to several files.
'dsn4' may be specified as for 'dsn2'

INIT(dsn5) Specify name of initial edit command file.
Commands from this file are obeyed before any commands are obtained from the terminal or WITH file. This is useful for changing the ECCE defaults to some user-selected values, or for defining frequently used macros.
'dsn5' may be specified as for 'dsn2'
INIT and OBEY are mutually exclusive.

OBEY(com) Specify actual initial commands to be obeyed before any commands are obtained from the terminal or WITH file.
INIT and OBEY are mutually exclusive.

ASIS/CAPS
EDCC/NOCC
HUSH
SCAN
SHOW
Options as described in Appendix B.

DCB(dcb) If a data set is being created then this specifies the model DCB to be used. For a list of valid DCB's use the TSO command AUDIT USER(DCB)
DCB and LIKE are mutually exclusive.

LIKE(dsn6) If a data set is being created then this specifies a data set to use as a model for DCB information.
'dsn6' may be specified as for 'dsn2'
DCB and LIKE are mutually exclusive.

SIZE(n) If a data set is being created then this specifies the initial amount of space (in tracks) to be allocated.

INC(n) If a data set is being created then this specifies the size (in tracks) of each extra extent to be allocated as the data set grows beyond its current size.

DIR(n) If a partitioned data set is being created then this specifies the number of directory blocks to allocate.

ERRCODE(n) Specify the condition code to be returned by the

ECCE command in the event of an error. Must be a number. This is only for specialised use.

TRACE(n) Specify trace level for the ECCE command. This is normally for specialised use, but TRACE(1) may be tried by the curious without any problems. Sensible values are as follows:

- 0 suppress all tracing (default),
- 1 report data set names and manipulation,
- 2 display all commands (after substitution),
- 3 display all commands (before substitution).

VERSION(n) Specify version of ECCE to be used. This is only for specialised use, and no responsibility is accepted for any consequences of using other than the default (you might access a version under test).

APPENDIX E Using ECCE Via Batch Job Control Language

APPENDIX E Using ECCE Via Batch Job Control Language

ECCE can usefully be used as an editor in a batch job, though extra care may be needed to avoid mistakes. The following points should be noted.

If an editing command fails in batch mode then ECCE will report this and will then abandon editing as if the command %A had been issued. (Interactive %A
editing is allowed to continue after a failure since it is assumed that the user can take some intelligent recovery action.)

If the end of the command file is reached then ECCE will report this and then abandon editing as if the command %A had been issued. The last line of the %A
editing command file should normally be %C (see Section 2.5). %C

The command file must not contain any prompts. All editing commands and inserted text are copied by ECCE from the command file to the report file, with the relevant prompts inserted, so that a user can see exactly what ECCE thinks it is doing. Also, the current line is normally copied to the report file after every command line to indicate what ECCE has done.

ECCE may be used in batch via Phoenix commands. The rest of this appendix applies when using batch JCL.

When ECCE is invoked directly by batch JCL statements the output file must be directly
distinct from the input file. The implicit use of temporary files available via TSO or Phoenix commands is not possible using batch JCL.

Simple example

Simple example

```
//(jobname) JOB  '(name,delivery instructions)'
//*PASSWORD (password)
//STEP1     EXEC ECCE,SRCE='(account no).(source)'
```

```
   //G.DEST    DD   DSN=(account no).(result),DISP=(NEW,CATLG,DELETE),
   //                UNIT=DISC,SPACE=(6000,(10,1),RLSE),DCB=DCB.FB
   //G.EDIT    DD   *
     ...
     editing commands
     ...
     %C
   /*
   //
```

where (jobname) is the user's account number optionally followed by a single capital letter,
       (password) is the user's password,
       (source)   is the name of the primary input file,
       (result)   is the name of the output file

For details of the DD statement and the SPACE and DCB parameters, refer to the MVS Handbook Chapter xx.

Syntax
Syntax

```
   //stepname EXEC ECCE,SRCE=dsname1,
   //             SECSRCE=dsname2,
   //             OPTION='opt1,opt2,...'
   //G.DEST  DD  ....
   //G.EDIT  DD  *
      ...
      editing commands
      ...
      %C
   /*
```

where opt1,opt2 are options as defined in Appendix B,
       G.DEST   may be omitted if option SCAN is used,
       SECSRCE   specifies the secondary input file.

ECCE return codes:

16	Pascal runtime failure (should not occur).
17	abandoned with %A command or after command failure
18	empty output file
19	unable to open required files

APPENDIX F Full-screen Editing

APPENDIX F Full-screen Editing

ECCE normally operates as a command-driven context editor. If ECCE is being used from a BBC micro-computer fitted with a DECCE chip, then it is possible to operate ECCE as if it were a full-screen editor. No special connection to the mainframe is needed. A fast line reduces delays. Full-screen mode is not suitable for files which contain lines more than 80 characters long. ((DECCE software may be available for other intelligent terminals such as IBM PC's. Note that all the full-screen manipulation is done within the intelligent terminal and that ECCE itself is unaware of what is going on!))

In full-screen mode, 24 lines from the file are displayed on the screen. Changes are made by moving the cursor to the required position and typing the required alterations, thus over-writing the original text. Cursor movement is controlled by the 'arrow' keys (referred to as: up,down,left,right) which are usually situated towards the right-hand side of the keyboard. Various control keys are used to scroll through the file, to delete or insert characters, and to perform other useful functions.

Full-screen mode is entered by keying CTRL/V (hold the key marked CTRL down and press the key marked V); context mode is re-entered by keying CTRL/C. Any number of switches between context and screen mode may be made during an editing session. If an attempt is made to move the cursor off the top or bottom of the screen, the file display is scrolled up or down 12 lines as appropriate. If an attempt is made to move the cursor onto the **END** line, a blank line is appended to the file.

Cursor movement

Cursor movement

left	Move the cursor left one character position. If the cursor is at the start of a line, it will move to the end of the previous line (scrolling the display down 12 lines if necessary).
right	Move the cursor right one character position. If the cursor is at the end of a line, it will move to the start of the next line (scrolling the display up 12 lines if necessary).
up	Move the cursor up one line (scrolling the display down 12 lines if necessary).
down	Move the cursor down one line (scrolling the display up 12 lines if necessary).
CTRL/F	Move the cursor left 6 character positions, if possible.
CTRL/K	Move the cursor left, to the start of the previous word, if possible. (SHIFT/left has the same effect).
CTRL/L	Move the cursor right, to the start of the next word, if possible. (SHIFT/right has the same effect).
CTRL/T	Move the cursor right 6 character positions, if possible.
CTRL/Z	Move the cursor right, to just after the last actual character on the current line.

Scrolling

Scrolling

CTRL/up	Move the cursor up 24 lines, scrolling as required.
CTRL/down	Move the cursor down 24 lines, scrolling as required.

CTRL/left Move the cursor up 12 lines, scrolling as required.

CTRL/right Move the cursor down 12 lines, scrolling as required.

CTRL/B Move cursor to beginning of file (c.f M-*). SHIFT/up has the same effect.

CTRL/E Move the cursor near to the end of file, with **END** in the middle of the screen, and the cursor located at the top left corner of the screen. SHIFT/down has the same effect.

Insertion, Line Joining and Splitting

Insertion, Line Joining and Splitting

CTRL/A Add the line below the cursor to the end of the current line.

CTRL/P Push text rightwards as far as possible by inserting spaces at the cursor. This allows text to be inserted at the cursor. (CTRL/Q may be then used to remove surplus spaces.)

CTRL/S Split line in two at the cursor position. Creates a blank line before the current line if the cursor is in column 1.

CTRL/W Push text rightwards one character position by inserting one space at the cursor. This allows a single character to be inserted. If this pushes text off the screen then the last character on the line will be replaced by tilde '~'. See CTRL/O below.

Deletion

Deletion

CTRL/D Delete current line from cursor position onwards. Delete whole line if the cursor is in column 1.

CTRL/Q Starting with the character at the cursor, remove spaces until a non-space character is at the cursor.

CTRL/R Remove the character at the cursor position.

Copying Text

Copying Text

CTRL/U Insert a line before the current line which contains a copy of the line buffer previously defined by CTRL/X.

CTRL/X Transfer a copy of the current line into a line buffer, replacing whatever is currently in the line buffer.

CTRL/COPY Toggle copy mode on or off. The 'arrow' keys normally move the cursor; in copy mode they may be used to copy text from one part of the screen to another by proceeding as follows:

a) move cursor to point at which text is to placed,
b) switch copy mode on using CTRL/COPY,
c) use 'arrow' keys to move cursor to text to be copied,
d) press COPY key as required until all text has been copied, repeat from (c) if text elsewhere is to be copied,
e) switch copy mode off using CTRL/COPY.

Miscellaneous

Miscellaneous

CTRL/C Switch to ECCE command mode. The ECCE pointer (@) will be at the position of the screen cursor.

CTRL/G Toggle word-wrap mode. Word-wrap mode allows continuous entry of text without needing to press RETURN (or equivalent). Whenever a word would overflow the right-hand edge of the screen it is removed from that line and placed at the start of a new following line.

CTRL/O Re-display the line on which the cursor is located. This function is used primarily when a line has previously been extended beyond the screen width and then shortened leaving tilde '~' characters representing the unseen display.

CTRL/V Switch to screen edit mode. The first line on the screen is the current line. **WINDOW TOP** indicates start of file and **END** indicates end of file. CTRL/V must be the only input in response to the primary input file prompt ')'. At the time of entry, the default line numbering display (restored by %P if necessary) must be in operation.

APPENDIX G Changes to ECCE Made in This Implementation

APPENDIX G Changes to ECCE Made in This Implementation

During the implementation of ECCE at ULCC a small number of user-visible changes have been made to the version supplied by Nottingham University ((and some internal errors have been corrected)). These changes are detailed below so that users may be aware of possible differences from other ECCE implementations.

(a) The space available to hold the internal copy of the edited file is much larger than in many other implementations of ECCE ((roughly 10 times larger)).

(b) The G* command can only be terminated by a line consisting of just a G* colon ':'. Other implementations treat as terminator any line that starts with a colon. (This change allows lines starting with a colon to be entered.)

(c) All query commands of the form %a? are new, where a = J,T,V,W,X,Y,Z.
J,T,V,W,X,Y,Z.
These commands allow a user to check on the current setting as defined by the last %a= command. Other implementations have no such facility and users must rely on their memory.

(d) Except for %J=n all variations of the %J= command are new. These
%J=n %J=
variations make it more convenient to display long lines or to suppress unwanted information such as sequence numbers in columns 73 onwards.

(e) The %R command has not been implemented due to technical difficulties.
%R
Most implementations which include %R have filing systems which fully support files with version numbers. This command is mainly useful when dealing with files which exceed the size of ECCE's internal buffer. At ULCC the buffer is much larger ((by roughly a factor of 10)) than those used elsewhere so few users will need %R.