University of Edinburgh

Department of Computer Science

ECCE
Edinburgh Compatible Context Editor
Video Version

by

Hamish Dewar

Internal Report

CSR-175-84

November, 1984

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

E C C E

E d i n b u r g h
C o m p a t i b l e
C o n t e x t
E d i t o r

v i d e o   v e r s i o n

Hamish Dewar
Department of Computer Science
Edinburgh University

The original design of ECCE dates from 1968 and its first implementations were for the DEC PDP-9 and the ICL System 4/75 computers. The principal aim of the design was to produce a context editor which would be powerful enough to be a convenient and flexible tool, but also simple enough to be readily implementable on a wide range of different systems. In this way it was hoped to make it easier for users to switch from one system to another by providing a familiar editing environment on each, and in the event ECCE has been implemented on more than a dozen different systems at Edinburgh and elsewhere. Extensions have, naturally, been made to suit local requirements, mostly in the spirit of the overall design philosophy.

Apart from the aim of remaining free of operating system and terminal device dependencies, some of the distinctive features of ECCE are:

- simple primitive commands with means of composition;
- uniform interpretation of numeric parameters;
- specification of a failure condition for each command;
- design of commands to be usefully repeatable;
- use of Regular Expressions to provide programmed commands;
- the Substitute command.

The need to revise several local implementations in order to take fuller advantage of the capabilities of video terminals has provided the opportunity to include in the basic specification some additional features which I and others have experimented with in particular versions.

I am grateful to a number of colleagues, past and present, who have contributed to the development of ECCE. Alan Freeman's paper-tape editor for the PDP-8 introduced me to the concept of context-editing and Chris Whitfield made many helpful comments on the original specification. More recently, Paul McLellan, John Murison, Douglas Buchanan, Richard Marshall and Graham Toal have been a source of ideas for improvement. Lee Smith was responsible for the first proper users' manual, which I have drawn on in preparing the present document.

Hamish Dewar

C O N T E N T S

1. INTRODUCTION
- -- text editing and text files 1
- -- the file pointer 1
- -- feedback - video and hard-copy 2
- -- editing modes - data-entry and command 3
- -- control keys 3
- -- learning to use the Editor 5

2. CALLING THE EDITOR
- -- system variations 6
- -- old and new files 6
- -- secondary input 7
- -- file-names and options 7
- -- example calling formats 8

3. COMMAND FORMATS
- -- types of command 9
- -- individual command formats 9
- -- case distinctions 10
- -- multiple commands 10
- -- syntactic errors 11
- -- failure conditions 11
- -- repetition numbers and command repetition 11

4. INDIVIDUAL EDITING COMMANDS
- -- list of commands 12

5. OPTIONS AND SPECIAL COMMANDS
- -- list of options 24
- -- environment command (%E) 26
- -- other special commands 27

6. COMMAND AND TEXT MACROS
- -- general 28
- -- enquiry facility (%Q) 28
- -- key definition facility (%K) 29

7. PROGRAMMED COMMANDS
- -- command sequences 31
- -- alternatives 31
- -- inverted failure condition 32
- -- cancelled failure condition 32
- -- example commands 33

8. COMMAND CHECKLISTS 34

1. INTRODUCTION

ECCE is a text editor. Under the control of commands entered from a terminal keyboard, it carries out modifications to a body of text held in a computer file. This text file may be, for example, a source program in some high-level language, or tables of results generated by a statistical package, or data intended as input to a user's program; it may simply be material which is held in the computer filing system solely for the purposes of editing and listing. The differences between these kinds of text files are not in any way significant to the operation of the Editor. What ECCE cannot do is edit files which are not composed of lines of text, such as executable code files.

A text file consists of a number of lines, each line containing a number of characters. The characters making up a line are either printing characters or space characters. When typing in lines on the computer terminal, RETURN is used to mark the end of the line. The Editor regards lines as being of variable length, usually ending with the last printing character. Hence 'white' space at the right-hand end of a line is not part of the line, in the way that space at the left or within the line is - being made up of individual space characters (corresponding to depressions of the space-bar). Although these make no impact on the printed page or video screen, they are just as significant for editing as other characters. For example, the line:

```
longing   for those  wide open      spaces
```

would look like this if the spaces were replaced by asterisks:

```
longing***for*those**wide*open******spaces
```

A blank line is an empty line; that is, it contains no characters, not even spaces. Blank lines are just as significant for editing as other lines.

The main function of the Editor is to enable the terminal user to make alterations to a file held within the computer system, but it may also be used to create a new file from scratch or simply to inspect a file without altering it.

The file pointer

The operation of the Editor is directed by a series of commands which are carried out one after the other in the order typed. The effect of most commands depends partly on the position of a notional pointer which identifies the 'current position reached' in the file. Initially the file pointer is positioned at the start of the first line in the file. It may be moved along lines and down from one line to the next. It may be positioned at the start of any line, between any two characters on a line, or at the end of a line. It may also be positioned at the very end of the file, that is, after the last line of text.

Many commands have no effect other than that of re-positioning the file pointer. The usual direction of movement is forwards, that is, from left to right along a line and downwards from one line to the next, but there are also commands which move the pointer backwards in the file. Commands which simply move the file pointer around will be called **location commands**; commands which actually change the material in the file will be called **alteration commands**. The typical pattern of use of the Editor is to go progressively through the file, using whatever location commands are most convenient to reach the next point at which a correction is required, and then use one or more alteration commands to effect the correction.

Feedback

ECCE commands are usually typed on-line, that is, in interactive mode. After the execution of each complete command, some feedback is provided to the user. This enables the user to confirm whether the command has had the effect intended, before issuing the next command.

The nature of the feedback depends on whether the Editor is being used from a hard-copy (printing) device or from a video terminal. As editors are increasingly being accessed in the latter way, the emphasis in most of this document is on that case. It is hoped that users of hard-copy devices (or videos used as hard-copy devices) will be able to ignore or re-interpret parts of the description which are not relevant to this mode of access.

Video feedback

In the case of videos, feedback is in the form of a 'window' displaying a group of consecutive lines in the file. As successive commands are executed, the window is updated so that it always includes the current line and so that the position of the pointer is indicated. Unfortunately video terminals vary considerably in terms of their capabilities and the way they are controlled. In consequence the fashion in which the window is updated and the pointer is displayed varies from terminal to terminal, and a particular implementation of the Editor will support only a particular selection of video terminals; others may have to be used as if they were hard-copy devices.

The size of the window used for video feedback varies between a minimum and maximum number of lines, within a screen region which may be specified by the user. Whenever editing moves to a fresh site in the file, the display is completely re-written, with a minimum size window. Moving off the bottom or top of the window causes the window to be extended, when this is possible.

A line which exceeds the window width is truncated rather than wrapped round on the display. The invisible characters can still be edited.

For purposes of printing or display, the end of the file is shown as an extra line, beyond the last genuine line, in the form:

```
        **END**
```

This line does not actually exist in the file and therefore cannot be edited.

Hard-copy feedback

In the case of printing terminals (or videos used as hard-copy devices), the feedback consists of printing out the current line, with the position of the file pointer indicated by insertion of a circumflex character (up-arrow on some terminals). However, this extra character is not included when the pointer is at the start of the line.

```
Thus:   here the file pointer is splitting ha^irs
and:    here it is at the end of the line^
but:    here it is at the start of the line
```

With slow terminals, it can be excessively time-consuming if the current line is printed out after every command is executed and there are options which may be selected to reduce the frequency of output.

Editing modes

There are two modes in which the Editor may be used from a video terminal, and the user may switch freely between them. They are data-entry mode and command mode.

In data-entry mode, the video cursor indicates the position of the Editor's file-pointer and text typed at the keyboard goes directly into the file. There are two varieties of data-entry mode: **replacement**, in which the new material overwrites the existing text, and **insertion**, in which the new material is inserted into the existing text. The only keys which have a command siginificance in data-entry mode are the control keys described in the following section.

In command mode, the full range of ECCE commands is available, not just the limited number available through control keys. In this mode, the Editor prompts for input in the command region of the screen and interprets the text typed as an editing command.

> Since, in this mode, the actual video cursor is pre-empted for use in the command region of the screen, a variety of alternative means are used to indicate the position of the Editor's file pointer within the file region. This may be in the same form as the video cursor but the choice is limited by terminal capabilities. Often reduced or increased intensity has to be used; in this case, a vertical bar is displayed if the character to the right of the pointer is a space or the pointer is at the end of the line. For terminals lacking the capability to highlight an arbitrary character position in any way, the pointer is shown as a circumflex (or other marker character) temporarily overwriting the character to the left of the pointer, and an extra column is added at the left of each line for use when the pointer is at the start of the line.

Control keys

It is part of the philosophy of ECCE that it should be possible to enter all the editing commands from any terminal, using just conventional printing characters and the line-terminator RETURN. No assumption is made about the availability of special control keys on the terminal keyboard. Accordingly all ECCE commands are represented by printing characters, which can be entered in command mode.

However, where a terminal is fitted with special keys, such as cursor controls, function keys, or a switchable numeric pad, these keys are usually a more convenient way of entering frequently required commands. Alphabetic keys used in conjunction with the CONTROL shift available on most terminals may also be used in this way. All such keys, or key combinations, will be referred to as control keys. The particular advantage of control keys is that they are immediate in their effect; they do not require a following RETURN.

Technically, these keys are macros. Macros are described in detail in Chapter 6 but the basic principle is that a single key can be defined to stand for an arbitrary sequence of characters. Initial definitions for some of the keys are pre-set on entry to the Editor, and others can be defined by the user, as required. This is part of a general capability to extend the repertoire of the Editor by means of user-defined commands. Sets of definitions can be saved in command files for subsequent use.

Because of the fact that the interpretation of control keys is not fixed and because of the considerable variation in what is available on different terminals, it is not possible in this document to provide an enumeration of their functions, despite the fact that they often provide the most convenient access to certain editing facilities.

Some typical examples of control key assignments are:

RETURN, used by itself, - Move to the next line in the file
Cursor arrow keys - Cursor Up, Cursor Down, Cursor Left, Cursor Right
LF key - repeat last command
TAB key - move to next word

One function which, as an exception to the general rule, cannot be invoked except by a control key is that of switching between command mode and data-entry mode. The restriction is solely to ensure that if data-entry mode can be entered, it can be left, since in data-entry mode, none of the printing characters has a command significance. This control key is referred to as the mode-switch key.

It should be noted that the possibility of using control keys at all depends on appropriate support being available in the terminal handling software. They may not be usable in all modes of connection, in particular across some communication networks. In addition, some control keys have a reserved significance for the operating system and it is advisable to find out what these are sooner rather than later.

The key enquiry command (%Q) which is described in Chapter 6 provides a means of investigating the significance of both control and ordinary keys.

Learning to use the Editor

To begin with, the Editor will seem a clumsy tool, particularly for those with limited typing experience. Changes which could be marked up on a document in no time take an age to translate into sequences of editing commands to achieve the desired effect. Fluency comes with practice, although editing remains a fiddling business, and mistakes are inevitable. Happily most mistakes are easily recovered from, and in the exceptional case where an edit goes disastrously wrong (for example unintentionally scrambling half the file), the editing session can always be abandoned without losing the original file, so that only the editing time is wasted.

With experience, it becomes quite straightforward, and habitual, to edit a file 'sight unseen'. However, for initial learning, it is sensible to use as test data a file with known contents. The following procedure is a possible learning programme:

(a) Beg or borrow a copy of some text file for experimentation.
A suitable length is 30 to 100 lines of text.

(b) Obtain a printer listing of the file.

(c) At the terminal give the appropriate command to call the Editor to edit the test file (see next chapter).

(d) Use the %Q command to explore the significance of the individual keys, particularly any control keys.

(e) Initially use only the location commands to move about the file without altering it in order to gain familiarity with the effect of these commands in detail.

(f) Mark up the listing of the file with a few typical 'corrections'.

(g) Move back to the top of the file and start applying the corrections one by one from beginning to end.

(h) When all the corrections have been applied, close off the edit.

(i) Obtain a printer listing of the revised file for re-assurance that the corrections have really taken effect.

There are quite a number of individual commands in ECCE, and each has some utility in particular cases. But there are only seven or eight which have a high frequency of use and indeed all editing can be done in terms of these. To begin with it is recommended that attention is confined to the following commands:

Move and Move Back	to move about the file
Cursor moves	to move about locally
Get and Kill	to insert and delete complete lines
Insert and Erase	to insert and delete within a line
Find	to find a text string
Substitute	to change a found text string

When these have been mastered, others can be added to the repertoire as the need for them is felt. There are often several ways of achieving the same effect with the Editor. The experienced user usually chooses one that involves least typing, but agonising over the choice can waste more time than is saved.

2. CALLING THE EDITOR

Obviously, ECCE is only one among many utility programs which may be used or called on a given system. The way in which this is done depends both on the particular system and on the implementation of the Editor. The system documentation will provide information about how to call programs and about the filing system, in particular the conventions for naming files. Beyond this, what the intending user needs to find out is:

whether any preliminary commands have to be given to make the Editor available for the first time

the name by which the Editor is known (maybe "ECCE", maybe just "E")

how to indicate in calling the Editor what file or files it is to operate on and what options are to be selected.

The examples given below are valid for a number of implementations, but details may vary in other systems, and are liable to change from time to time.

Once the Editor has been called, the commands described in the rest of this document are available for use. To terminate an editing session in normal fashion - that is, with all changes made permanent in the file specified as output - the Close command %C (or control key equivalent) is typed.

Old and new files

While it is natural to speak of using the Editor to alter or modify a file, it is important for some purposes to understand that the Editor does not literally modify an existing file. What it does is produce a completely new file from the old one. Each editing session creates a fresh file, though in most cases the bulk of the material in it is copied verbatim from the old one. At the conclusion of an editing session, the new file created supersedes the old one, so that the file-name which previously designated the old file now designates the new one.

In this typical case, the effect is similar to what would happen if the Editor did actually alter the old file. However, the following consequences indicate why it is necessary to make the distinction:

if an editing session is abandoned, rather than being terminated normally, no new file is created and the old file remains unaltered;

the Editor can be called in such a way that the new file is given a different name from the old file so that the old also continues in existence at the end of the editing session;

on systems which automatically preserve earlier generations of like-named files, the old file also survives (though an earlier version may be purged).

Secondary input files

The requirement often arises to incorporate into one file part or all of another file. Many computer systems provide a command to handle the particular case of concatenating complete files, either through the provision of a special utility program or as an option within a Copy or Transfer utility. The secondary input facility in ECCE provides a more general capability.

In general terms the facility permits the user to switch from the main file to an alternative file and there select portions of that file to be incorporated into the file being edited.

A file to be used for secondary input can be nominated at the time the Editor is called, along with the old and new file-names. Such files can also be specified in the course of the edit, by means of the %S command.

Options

It is possible when calling the Editor to select a number of options which control the way in which the Editor operates, for example, what size of display window it uses. Some option parameters have numeric values (such as the height of the window) and others are simply selectors (such as whether case-distinctions are to be treated as significant when matching letters of the alphabet). Options are introduced by a qualifier symbol and a keyword; for a parameter with a variable value, this is followed by an equals-sign and a value. Examples of complete option specifications, assuming the dash (minus) as the qualifier symbol, are "-MINWIN=12" and "-NOMATCH". Some of these options may also be varied during the course of the edit by means of the Environment command %E. Fuller information about options is given in Chapter 5.

Example calling formats

The examples which follow are valid system commands on several implementations of ECCE. They pre-suppose that the Editor is known as "E" so that each command starts with "E" followed by a space, and then the name(s) of the file(s) to be operated on. On these systems an oblique stroke is used as a separator between input file names and output file names.

(a) To edit an existing file called "CURTEXT" so that the new file created will also be called "CURTEXT":

```
E CURTEXT
```

This is the standard updating procedure.

(b) To edit an existing file called "OLDTEXT" in such a way as to create a new file called "NEWTEXT":

```
E OLDTEXT/NEWTEXT
```

(c) To create a new file called "PROG34" from scratch:

```
E .N/PROG34
```

Technically this is a special case of (b) above starting with an empty old file. The Editor always accepts ".N" to indicate a null file, but usually also accepts the system standard designation for such a file, if there is one. The first editing command when creating a new file would normally be G* (see Get below)

(d) To inspect an existing file called "RESULTS" without altering it:

```
E RESULTS/.N
```

Again this is a special case of (b) with a null new file. Alteration commands are dis-allowed in this mode of operation.

(e) To edit an existing file "PROG" together with a secondary input file "SPECS" in order to create a new file "FULLPROG":

```
E PROG,SPECS/FULLPROG
```

(f) examples including option specifiers:

```
E CURTEXT-MINWIN=12
E CURTEXT-NOMATCH-PRE=EDCOM1
```

In all cases where reference is made to a new file being created, the effect is to supersede any existing file of the same name,, whether that file has been specified as an input to the Editor or not. On systems which do not automatically maintain several generations of a file, the existing file is lost.

3. COMMAND FORMATS

When the Editor is called, it positions its file pointer at the start of the first line in the file to be edited and presents the initial display.

In data-entry mode, only control keys (including RETURN) are interpreted as commands. Any printing characters and spaces typed are treated as data to go into the file. In the replacement variant of data-entry mode, the characters typed overwrite the existing text on a one-for-one basis. In the insertion variant, characters are successively inserted into the text to the left of the file pointer. The mode-switch control key is used to go from data-entry mode to command mode, and conversely. However, if it is pressed twice in succession (that is, no command is typed in command mode), it switches between the two variants of data-entry mode.

In some implementations, it may not be feasible to provide the insertion capability, so that only the overwrite capability is available

In command mode, the Editor prompts for input in the command region of the window. The response may take one of three forms:

typing a command line followed by RETURN
typing a Special command followed by RETURN
pressing a control key

A command line consists of one or more editing commands, optionally separated by spaces. A command line is terminated by RETURN and errors noticed while typing it may be cancelled by means of the DEL key, and any other universal line-correction facilities provided by the system.

A Special command is distinguished by starting with one of a number of prefix characters. The prefix % (percent-sign) is used to introduce a Special command within the Editor's defined repertoire; the prefix ! (exclamation-mark) is used to introduce a system command to be called from the Editor. (Note that this interpretation applies only when the character appears at the start of a command). Only the percent-sign commands are described here, since the others are, by definition, system-dependent. These commands consist of a percent-sign followed by a letter and are used to select options and set modes which modify the effect of subsequent editing commands. The most essential Special command is the Close command %C which is used to close off the edit. In many implementations, the Special command %H (for Help) is available, to provide general Help information on the Editor and its use (see also the %Q command). The remaining Special commands of this type are described in Chapters 5 and 6.

Pressing a control key which is defined as a command sequence has the same effect as if the the user had typed in the command sequence followed by RETURN. Note, however, the important difference mentioned under 'repetition'. RETURN by itself counts as a control key with the significance of M (Move).

Individual editing commands

There are just two formats for individual editing commands, one for those commands which are accompanied by a text string as parameter and one for those which are not.

A command which has no parameter takes the form of a single character or a single character followed by a minus sign. For most commands the single character is a letter, which is mnemonic for an imperative verb. For example, the Move command is denoted simply by M (or m) and the Move Back command by M- (or m-).

A command with a text parameter takes the form of a single letter (or letter followed by minus) followed by whatever string of characters the user wants to specify, enclosed within delimiter characters. For example, F/cat/ is an instance of the Find command with cat as its parameter, and S.dog. is an instance of the Substitute command with dog as its parameter. The user has a choice of several characters for use as delimiters; the oblique stroke and the period illustrated above are popular because of typing convenience, but any character which has no defined significance to the Editor may be used. The opening and closing delimiter for any one parameter must be the same, and the delimiter must be chosen to be distinct from any of the characters which require to be included in the text string. Strings may include any printing characters and spaces, but not line-breaks.

As well as quoted strings, there are a number of other ways in which a text parameter may be specified:

by one of the three text macro letters (upper-case X,Y,Z);
by the ditto symbol ("):
by the exclamation-mark (!).

The use of X,Y,Z is described in Chapter 6.

The ditto symbol indicates that the text string to be used is the same as the last string used in a command of the same group (matching or insertion).

Using an exclamation-mark in place of a text string indicates that the actual text to be used is not provided within the command, but is to be requested at the time the command is executed. In this case, when the text is requested, it should be typed without delimiters and terminated by RETURN. (For the interpretation of exclamation-mark within command macro sequences, see Chapter 6).

The commands which take a text parameter fall into two groups: text-matching commands and text-insertion commands. For typing convenience, there is some relaxation of the format rules for the parameters of the latter category (Insert, Overwrite, Substitute and Get). The closing delimiter for quoted text may be omitted if the text string is the last thing in a command line, and the exclamation-mark indicating the direct-entry case may also be omitted, so that just leaving out the parameter indicates that the text is to be requested at the point of execution.

Case distinctions

In commands, the upper-case letters A to W have a fixed significance as basic editing commands. Initially the lower-case letters a to w have the same meaning as their upper-case counterparts, but this is not fixed and may be changed by re-definition as command macros. The letters X to Z and x to z are distinct, the former being available for definition as text macros and the latter as command macros.

Within text strings, case distinctions are always significant for insertions, but are normally ignored for the purposes of text matching, so that, for example, the string "the" would be considered to match "The" as well as "the" (not to mention "tHe" and so on). However, there is an option to select that case distinctions are to be treated as significant for matching.

Multiple commands

When more than one command is typed on a line, they may be, but need not be, separated by spaces. The commands are executed in order from left to right. One point of putting several commands on one line is to control the frequency of feedback. Another is to produce a useful unit for repetition.

Syntactic errors

Before proceeding to execute a command line, the Editor checks that the format of each command in it is correct; for example, that a parameterless command is not followed by quoted text. If the command syntax is faulty, an error report is made indicating the nature of the error and no attempt is made to execute any of the commands making up the line. Note the contrast between the treatment of this kind of error and the effect of failures in executing well-formed commands described in the following section.

Failure conditions

The following pages provide a detailed description of the effect of the various commands. This defines not only what happens when the command is carried out but the condition under which the command cannot be carried out – the failure condition. In the simple case, a failure in command execution counts as an error and a report is made. Any subsequent commands in a command sequence are not executed, but the effect of any earlier commands in a command sequence is not undone, nor that of successfully completed cases of a repeated command. There may also be partial execution of the failing command itself, as defined in the individual descriptions which follow.

However, failure conditions can be utilised to control command execution in a number of ways, of which the most important is mentioned in the next section.

For a few commands, mainly those represented by a character other than a letter, there is no failure condition.

Repetition numbers

Any command may be followed by a repetition number (a decimal integer) indicating that the command is to be repeated the number of times specified. For example, M5 means Move five times, and I/ /20 means Insert a space twenty times, that is, in effect, insert twenty spaces. An asterisk may be used in place of a repetition number with the significance: repeat the command until it fails. For typing convenience, the digit zero may be employed in place of an asterisk. For example, the command E* (or E0) means Erase repeatedly until the failure condition for Erase is met, that is, Erase up to the end of the line.

Repetition of last command

Instead of typing in a new command at any point, the user may instead type simply a repetition number. This causes the last genuine command line to be executed again the number of times specified. The whole command line, which may consist of a sequence of several commands, is repeated. In this context, neither Special commands nor control key commands count as genuine command lines, nor do they cause the last genuine command to be lost (although some Special commands may cause the record of it to be erased from the screen).

M Move (forward one line)

The Move command causes the file pointer to be moved from its current position to the start of the following line.

Attaching a repetition number to a Move command provides a means of moving forward a fixed number of lines. For example, the command M99 issued at the start of the file, causes the pointer to be moved to the start of the hundredth line in the file.

The command fails if the file pointer is at the very end of the file, that is, beyond the last line of text.

It is legitimate to Move from the last line in the file to the end-of-file position, but then further Moves will fail.

M- Move Back (one line)

This command causes the file pointer to be moved from its current position to the start of the previous line.

The command fails if the current line is the first line in the file, the pointer being moved nonetheless to the start of this line.

In some implementations in which only a limited part of the file can be retained for editing, the Move Back command also fails if an attempt is made to go back past the start of the retained section.

n Move to line number *n*

This command causes the file pointer to be moved from its current position to the start of the line specified.

Lines are numbered from 1 at the start of the file and the numbers relate to the current state of the file, not its state at the start of editing. If no line number is specified in the command, one is requested at the time the command is executed.

The command is executed as a Move or Move Back as appropriate, and fails as for Move if the line number is too big.

Note that the parameter *n* is not a repetition count.

} (right brace) Cursor Down

This command is similar to the Move command, except that the pointer is moved to the column position in the following line corresponding to its position in the current line.

The command is typically the definition of a cursor control key. With this and the other cursor movement commands, the resulting line position may be beyond the end of the line (that is, to the right of the last printing character). This is often a temporary state en route to another position, and so the fact that the cursor is out in space does not of itself imply any extension of the line. However, if any text is entered in this situation, the gap between the existing end of the line and the file pointer is first filled with spaces.

The command fails when the file pointer is on (or beyond) the last line of the file.

{ (left brace) Cursor Up

This command is similar to the Move Back command, except that the pointer is moved to the column position in the previous line corresponding to its position in the current line.

See the notes on Cursor Down

The command fails when the file pointer is on the first line of the file, and the pointer does not move.

R Right-shift (one character position)

This command causes the file pointer to be moved right one character position.

The command fails if the file pointer is at or beyond the end of the current line.

Hence R* takes the pointer from anywhere within the line to the right hand end of the line.

L Left-shift (one character position)

This command causes the file pointer to be moved left one character position.

The command fails if the file pointer is at the start of the current line.

Hence L* takes the pointer from any position to the start of the line.

> Cursor Right

This command is similar to the Right-shift command except that it permits the file pointer to be moved beyond the end of the line, that is, beyond the rightmost character on the line.

See the notes on Cursor Down.

The Cursor Right command fails if the file pointer is at or beyond the maximum line length defined by WIDTH.

< Cursor Left

The Cursor Left command is identical in effect to the left-shift command. It is provided for reasons of symmetry.

F/.../ Find TEXT

The Find command causes the Editor to search forward in the file for the first occurrence of the specified text string. The search starts at the current position of the file pointer, except that an occurrence just at the file pointer which has already been located, either by Find or by Uncover or Verify, is ignored.

> For example, adding a repetition count to a Find command, as in F/cat/3, will locate the third occurrence of the sequence cat. The repetition does not have to be in the form of a count attached to the original command. For example, it is a common experience to discover, after a Find has been executed once, that the character sequence chosen in fact appears earlier in the file than the intended position. In this case the Find command can simply be repeated, by typing I as a repetition command, until the desired occurrence is reached.

By default the search continues to the end of the file. The scope of the search may, however, be limited to a particular number of lines counting from (and including) the current line, by specifying the number of lines between the command letter and the quoted text.

> For example, F9/cat/ limits the scope of the search to the rest of the current line and the next eight lines.

The command fails if there is no occurrence of the specified sequence of characters within the scope of the search. Where the scope is more than one line, the file pointer is moved to the start of the last line searched (end-of-file in the case of unlimited scope). Where the scope is one line, the file pointer is not moved.

F-/.../ Find Back TEXT

The Find Back command is similar in effect to Find except that the search proceeds backwards through the file. The number of lines to be searched may be specified between the minus and the text parameter; by default the search continues to the start of the file.

The command fails if there is no occurrence of the specified text within the scope of the search, the file pointer ending up at the start of the last line searched.

T/.../ Traverse TEXT

The Traverse command causes the first occurrence of the specified text to be located and the pointer to be positioned at the right of the occurrence, rather than at the start as with Find. The search proceeds as for Find except that the default scope is confined to the current line and that an occurrence of text already located is not ignored.

The command fails if there is no occurrence of the text string within the scope of the search. Pointer movement in the case of failure is exactly as for Find.

V/.../ Verify TEXT

The Verify command neither moves the file pointer nor alters the content of the file. It succeeds if the sequence of characters immediately to the right of the file pointer matches the text specified; otherwise it fails.

The Verify command only has a use in conjunction with programmed commands (see Chapter 7).

I/.../ Insert TEXT

This command has the effect of inserting the text specified to the left of the file pointer. The text parameter which follows the letter I may take any of the forms described in Chapter 3, that is:

quoted text between delimiter characters;
text macro letter X,Y,Z;
the ditto character;
the exclamation-mark (or null).

The last-mentioned case I! provides a direct-entry capability for inserting a text string within a line. The text to be inserted is requested when the Insert command is executed, with the video cursor at the position of the file pointer. The normal way of terminating the text is with RETURN; if any other control key is used as a terminator, it causes the command sequence currently being executed to be abandoned and the control key to be treated as a new command. This applies also to the use of this form for the other insertion commands (Overwrite, Substitute, and Get).

For the direct-entry form, in most implementations, a certain amount of space is opened up at the position of the file pointer to permit the text to be typed in. Then when the terminating key is pressed, the line is closed up again. In other implementations, no space is opened up, and the line automatically accommodates each character as it is typed.
Specifying a repetition count with Insert is a convenient way of inserting multiple characters, typically spaces.

Insert fails if adding the text would cause the part of the line to the left of the pointer to exceed the maximum line length (as defined by WIDTH).

The failure condition is chiefly a precaution against unintended repetition of an Insert command.

O/.../ Overwrite with TEXT

The Overwrite command provides a means of replacing existing text on a one-for-one basis by new text. It differs from Insert in that one character is deleted from the file for each character added, except that if the end of the line is reached, it functions identically to insertion.

The possible forms of text parameter are as for Insert. The exclamation-mark form (O! or just O) again provides a direct-entry capability for a single piece of text.

The failure condition is the same as for Insert.

S/.../ Substitute TEXT

The Substitute command causes the text matched by an earlier Find (or Uncover or Verify) to be deleted and the text specified as parameter following the S to be inserted.

The operation of substitution involves two text strings, one to be removed, the other to be inserted. In ECCE the first of these is established by one command, typically a Find, and the second is specified as the parameter to Substitute. Thus, for example, F/this/ S/these/ will alter the first occurrence of this to these. One advantage of splitting the function between two commands is that they can be issued independently and the effect of the Find checked before giving the Substitute.

Substitute fails if the last positioning action was not a Find, Uncover or Verify, or the effect of inserting the text would exceed the maximum line length permitted by the Editor.

G/.../ Get TEXT (as complete line)

The Get command causes the text specified to be inserted as a complete line above the current line. The file pointer is moved to the start of the current line if it is not already there, but this remains the current line.

The Get command is the way in which complete lines are inserted in a file. The most frequently used form is G! (or simply G). In this case the text is not embedded in the command as a text string, but is requested when the Get is executed by the Editor. In this case, a repeated Get expects a fresh line each time; it does not cause the first line typed to be inserted again. Thus G5) causes five lines to be requested and inserted, in the order typed, above the current line.

The Get command fails if the line typed starts with a colon.

The failure condition is an arbitrary convention, which is used to cause a repeated Get to terminate. In particular, it permits the use of G*) to insert an indefinite number of lines (which it would be inconvenient to have to count).
An obvious consequence of the convention is that it is not possible using Get to insert a line of data starting with a colon. Ingenuity will suggest a variety of ways in which this restriction may be overcome if the situation should arise (for example, typing a space in front of the colon and removing it later).

K Kill (complete line)

The Kill command causes the whole of the current line to be deleted. The file pointer is left positioned at the start of the following line.

Kill does not simply erase all the characters on a line leaving it blank; it removes the line altogether. Kill followed by Get (eg KG or K2G3) is a common command sequence when replacing a number of complete lines by other complete lines.

The command fails if the file pointer is at the end of the file.

K- Kill Back (complete line)

This command causes the whole of the line before the current line to be deleted. The file pointer is moved to the start of the current line.

The command fails if the current line is the first line in the file, the pointer being moved nonetheless to the start of this line.

D/.../ Delete TEXT

The Delete command causes the first occurrence of the specified text to be located and then deleted, leaving the file pointer at the site of the deletion. The search proceeds as for Find, except that the default scope is confined to the current line, rather than the rest of the file, and an occurrence of text already matched is not ignored.

The command fails if there is no occurrence of the specified text within the scope of the command. Pointer movement in cases of failure is exactly as for Find.

E Erase (one character)

This command causes the character immediately to the right of the file pointer to be deleted

The command fails if the file pointer is at or beyond the end of the current line.

Hence E* erases all characters on the line to the right of the pointer.

E- Erase Back (one character)

This command causes the character immediately to the left of the file pointer to be deleted

The command fails if the pointer is at the start of the line.

Hence E-* erases all characters to the left of the pointer.

C Case-change (one character)

If the character immediately to the right of the file pointer is a letter of the alphabet, the Case-change command alters it to the corresponding letter in the other case, so that upper is mapped to lower and lower to upper. Whether the character is a letter or not, the pointer is moved one position to the right.

The command fails if the pointer is at or beyond the end of the current line.

Hence C* changes the case of all letters to the right of the pointer on the current line.

C- Case-change Back (one character)

If the character immediately to the left of the file pointer is a letter of the alphabet, the Case-change Back command alters it to the corresponding letter in the other case, so that upper is mapped to lower and lower to upper. Whether the character is a letter or not, the pointer is moved one position to the left.

The command fails if the pointer is at the start of the current line.

U/.../ Uncover TEXT

The Uncover command causes the first (or next) occurrence of the specified text to be located and all the material between the position of the file pointer at the start of the command and the occurrence of the text to be deleted. The text itself is not deleted. The search proceeds as for Find, except that the default scope is confined to the current line.

The Uncover command is applicable where it is most convenient to specify a deletion as 'up to' a particular character or sequence of characters. With an explicit scope, Uncover can delete part of the starting line, a number of complete lines, and an initial part of the line in which the string is located.

The command fails if there is no occurrence of the text string within the scope of the search. In the case of the default scope (current line only), the pointer is not moved and the line is not changed. In the case of a multi-line search, failure results in deletion of all material between the initial position of the pointer and the start of the last line searched, which is where the pointer ends up.

An unlimited search can be specified by means of U*.

B Break (current line in two)

This command causes the current line to be split in two at the position of the file pointer; the second part becomes the current line.

Break creates two lines from one. One of its uses is to split lines which may have become undesirably long as a result of insertions.
Break with the pointer at the start of a line creates a blank line above the current line.

There is no natural failure condition for this command; implementations generally impose an upper limit on the number of times it may be repeated.

J Join (delete line-break)

This command causes the following line to be appended to the current line, that is, it creates one line from two. The pointer is moved to the position of the join.

The command fails if the current line already exceeds the length defined as the maximum line width. In the case of failure the pointer is positioned at the end of the current line.

I- Insert Back
G- Get Back

The commands I- and G- re-introduce material which has been deleted from the file, the insertion being made at the position at which they are executed, which may or may not be the position at which the material was deleted. The order of recovery is the reverse of the order of deletion, and, in tune with this, these commands leave the file pointer in front of, rather than after, the newly inserted material. The use of these commands at a different site from the one at which the material was deleted is the basic technique in ECCE of moving blocks of text from one position to another.

The command I- restores a single deleted character from the last alteration site in the file. The last alteration site is the position in the file at which a sequence of contiguous insertions and deletions was last carried out. All the material deleted at this site is eligible for insertion. The command fails if there are no remaining deleted characters from that site.

For example, the command I-* brings in at any point all the text deleted at the last alteration site; and the sequence ERI- (Erase, Right, Insert Back) reverses the order of a pair of characters.

The command G- restores a complete deleted line. All complete lines deleted from the file at any time in the course of the edit are available for recovery, not just those removed at the last alteration site. It fails if there are no further deletions to be restored.

For example, KMG- (Kill, Move, Get Back) reverses the order of a pair of lines.

O- Overwrite Back

The command O- is an 'undo' command which can both restore material just deleted and remove material just inserted. It operates at the last alteration site in the file, irrespective of the position of the file pointer at the time the O- is executed. The last alteration site is the position in the file at which a sequence of contiguous insertions and deletions was last carried out. All the material deleted at this site is eligible for re-insertion, and all the inserted material is eligible for removal.
Each execution of the O- command removes a single inserted character or line break and/or restores a single deleted character or line break. It fails if neither of these operations is possible. Hence O-* removes the effect of all the alterations at the site.

P Print

The Print command is used solely to achieve feedback from the Editor. It causes the current line to be displayed, and, if repeated, causes a Move to the following line.

N Next word / bracket

This command locates the next word or matching bracket in the file, depending on the character at the starting point.

If the character currently to the right of the pointer is not a left bracket, the pointer is moved to the beginning of the next word, where a word is defined as a sequence of alphanumeric characters (letters or digits) not preceded by an alphanumeric character.

If the character is a left bracket (round, square or curly), the pointer is moved to the corresponding right bracket, taking account of nested occurrences of bracket symbols.

The file pointer ends up at the start of the word or bracket located, which is regarded as having been matched by a text search (so that Substitute is valid, for example).

As with Find, an already matched text sequence is ignored.

The command fails if there is no word or matching bracket before the end of the file.

N− Next word/bracket Back

This command locates the previous word or matching bracket in the file, on a similar basis as for Next, with the roles of left and right brackets reversed.

It fails if there is no word or matching bracket before the beginning of the file.

Q Query Spelling

The Query Spelling command causes the spelling of the next word in the text to be checked. This is done by consulting a spelling dictionary.

The command fails if the check fails. Repeated execution implies advancing to the next word for checking.

Hence the command Q* causes the check to be performed on successive words until one fails.

+*n* Increment number by *n*

The Increment command (plus-sign) is used to add the value specified by *n* to the first number to the right of the pointer on the current line. For this purpose, a number is any sequence of decimal digits, or a single letter.

This command is most useful when it is required to adjust more than one number by a constant amount. The value 1 is understood if *n* is omitted. The form +− may be used to specify a negative increment.

Letters are included to cover the case where they are used to number paragraphs.

The command fails if there is no number to the right of the pointer on the current line.

Note that the parameter *n* is not a repetition count.

@n At column position *n*

The purpose of the 'at' command is to enable text to be aligned to a particular column position. The effect is that the part of the current line to the right of the file pointer is aligned to the column position specified by *n*. This is achieved by the insertion or deletion of spaces to the left of the file pointer. In the event that aligning to the column position specified would cause the length of the line to exceed the defined maximum line length, *n* is reduced to prevent this.

Columns are numbered from zero upwards, so that, for example, @40 has the natural interpretation of placing text half-way along an 80-column line. Spaces to the right of the file pointer are not affected by this command, and only space characters immediately to the left of the file pointer are removed in seeking to align leftwards.

Using @ with a large *n* has the effect of right-aligning the text following the pointer.

The @*n* command fails if space-deletion fails to achieve correct alignment.

Note that the parameter *n* is not a repetition count.

A Adjust line length

The Adjust command is provided to simplify the task of maintaining a reasonably uniform line length within running text. In brief, it has the effect of breaking over-length lines and extending under-length lines by bringing up words from succeeding lines. The line length applied is as defined by the parameter WIDTH, and the left margin as defined by MARGIN

The detailed specification which follows is quite complicated. The main points to note are:
a single Adjust will make the current line an acceptable length and leave the file pointer at the start of the (perhaps revised) following line:
A* will produce adjusted lines up to the end of a paragraph, a paragraph being understood to terminate with a blank line (or end of file).

(a) If the end of the file has been reached, the command fails with no effect.

(b) If the current line (to the right of any margin) is blank, it is not changed and the file pointer is moved to the start of the following line.

(c) If the length of the current line exceeds the defined maximum line length, the line is broken at the rightmost space character to the right of the file pointer which leaves a line not exceeding the maximum length, and a new line is formed from the latter part of the original line, with an added margin of spaces if MARGIN is non-zero. The file pointer is left at the effective start of the new line. The command fails if there is no space character satisfying the condition specified.

(d) If the length of the current line is less than the maximum length, 'words' from succeeding lines are appended to the current line until either doing so would exceed the maximum length or a blank line (or end of file) is reached. The latter case is a failure condition, but in all cases the file pointer ends up at the start of the line following the adjusted line.

For the purposes of Adjust, a word is defined to be any sequence of characters terminated by a space or end of line. Appending a word implies inserting a space plus the word at the end of the first line, and removing the word plus its terminating space from the second line (removing the whole line in the case that the word is the only word on the line).

^

Set Marker

Execution of the Set Marker command (caret or circumflex) causes the current position of the file pointer to be noted for future reference.

The marker set by this command may be utilised by any of the following commands:

the Define Macro command
the Revert to Marker command
the Switch Inputs command

A marker is cancelled if the text surrounding it is deleted from the file.

^*n*

Define Macro

The Define Macro command (caret followed by a number in the range 1 to 6) serves to define one of the six macro letters X,Y,Z,x,y,z – with 1 corresponding to X and 6 corresponding to z. The effect is to define the selected letter to stand for the complete sequence of text between the set marker and the current position, or, if no marker is set, the text just matched by a text matching command.

The text sequence may extend over several lines, subject to an overall length limit imposed by the implementation. However, text sequences containing line breaks can be used only with the insertion commands, not the text matching ones.
It is immaterial whether the current position is after the marker or the other way round.

With a marker set, a second caret without an accompanying *n* is interpreted as Define Marker 1 (that is, as defining X).

This alternating significance of caret by itself is useful when it is set up as the definition of a control key.

=

Revert to Marker

The Revert command (equals-sign) has the property of restoring the file pointer to the position established by the last executed Set Marker. The marker is cancelled.

The command fails if there is no marker set.

|

Toggle Destruction Mode

Successive execution of this command (vertical bar) alternately enters or leaves a special mode of operation called destructive mode. In this state, any of the movement commands may be used in a destructive sense, that is in such a way as to delete all the material from the starting position to the position after executing the move.

Note that actual alteration commands are disallowed in this mode. The mode is most useful in conjunction with some of the more specialised movement commands such as Next, Cursor Down and Query. For example, a delete word command can be defined as |N| (Toggle, Next, Toggle), and a command to delete the rest of the current line and the beginning of the next line up to the current column position can be defined by |↓| (Toggle, Cursor-Down, Toggle).

The existence of this mode is indicated by a special prompt.

$ Switch Inputs

The Switch Inputs command (dollar-sign) is used to switch from the main file to the secondary input or back again. Having switched to secondary input, any of the location commands may be used to move about within that file, but alteration commands are not valid. The current position in the two files is independent and is preserved when a switch is made between the two. When this facility is used, a separate window is created out of the screen region used for file display, to show the current part of the secondary input file.

Apart from providing the facility simply to inspect another file while editing, the main use of secondary input is to enable material from the second file to be incorporated in the first. One way of doing this is to Define text macros to represent pieces of text in the secondary file and use these as parameters for Insert after switching back to the main file.

To provide a more convenient way of handling the most common requirements, the convention is also adopted that if a set marker is outstanding when a switch is made from secondary input to the main file, the text from the position of the marker up to the current position within the secondary file is immediately inserted in the main file.

For example, incorporating the whole of a secondary file in the main file can be achieved by moving to the appropriate point in the main file and then giving the commands (separately or as one command line):

```
$ ~ M* $
```

that is, Switch to secondary input, Set Marker, Move to end of file, Switch back to main file.

Any marker set in the main file is cancelled on switching to secondary input.

5. OPTIONS

This chapter describes the various options which may be selected to control or qualify the operation of the Editor. The list given includes those available in a number of implementations of ECCE; particular versions may have more or less extensive lists. The parameters described first and marked with an asterisk can be specified only in the command line when the Editor is called; that is, once established, they cannot be changed. The others may be freely altered at any time, by means of the Environment command described below.

*** PRE – Pre-definition file** **Default: null**

This parameter is provided to make it possible to specify a file of editing commands which are to be obeyed at the outset of the editing session. Its main use is to permit frequently used definitions to be recovered from a file, but the file may contain any editing commands and may in fact comprise a complete edit ending with the Close command %C. Values for this parameter must be valid file-names, for example -PRE=PASCALDEF. Although PRE may be specified at the outset only, the %G (Get commands from file) is available at any time.

*** TTYPE – terminal type** **Default: *see text*** **Range: 1–30**

The parameter TTYPE informs the Editor what type of interactive terminal is being used and hence determines the terminal characteristics of the device, including the screen length and screen width. Values for this parameter are numbers in the encoding used generally on the system (ERCC enumeration at Edinburgh, for example -TTYPE=7 for VT100). The default value is the system-defined TERMINALTYPE, and in general this should be appropriate.

*** WTOP – window top**	**Default: 0**	**Range: 0..VROWS–1**
*** WROWS – window rows**	**Default: VROWS–2**	**Range: 1..VROWS–2**
*** WLEFT – window left**	**Default: 0**	**Range: 0..VCOLS**
*** WCOLS – window columns**	**Default: VCOLS**	**Range: 1..VCOLS**
*** CTOP – command top**	**Default: WROWS**	**Range: 0..VROWS–2**
*** CLEFT – command left**	**Default: 0**	**Range: 0..VCOLS–40**
*** CCOLS – command columns**	**Default: VCOLS**	**Range: 40..VCOLS**

These parameters define the two screen regions (file and command) to be used by the Editor. The 'top' values are in terms of row numbers ranging from zero at the top of the actual screen to VROWS–1 at the bottom; VROWS is typically 24. The 'left' values relate to columns, numbered from zero at the left of the actual screen to VCOLS–1 at the right; VCOLS is typically 80.

The first four define the total extent of the screen region to be used for displaying the file being edited. The window height, as defined by WROWS, may be anything from one row up to the full screen height less two rows. A one row window obviously gives a rather blinkered view of the file. The width of the window, as defined by WCOLS, may be anything from one column to the full screen size. A one column window is obviously daft.

The parameters CTOP, CLEFT and CCOLS define the position and width of the two-row region used for commands and reports. CTOP specifies the first of the two rows. The minimum width for the command region, as defined by CCOLS, is 40.

Usually when the Editor is called directly by the user from system command level, it is appropriate for these two regions to occupy the full screen, but when the Editor is called from within another package or if the user wants to preserve other information on the screen, a smaller effective screen area may be specified. The Editor neither clears nor modifies areas outside the regions specified by these parameters.

With the default window values, the command region occupies the last two rows on the screen and the file window the remainder.

*** MAXWIN – maximum window** Default: WROWS Range: 1:WROWS
MINWIN – minimum window Default: ? Range: 1:WROWS

The purpose of these parameters is to make it possible to control the volume of text which is transferred to the screen at any one time to show the state of the file. The principle is that when the focus of editing moves to a completely new site in the file, only MINWIN rows are displayed in the lower part of the available window area. During local manoeuvres, this minimum display is extended as appropriate up to the number of rows specified by MAXWIN. (What operations permit extension depends on the video characteristics and the position and width of the window).

The MAXWIN parameter selects the amount of the file window which is to be used at the outset, the remainder of WROWS being reserved for secondary input.

When a secondary input file is being processed, it has its own varying size window at the top of the overall file display region defined by WTOP and WROWS. The size initially is determined by MAXWIN (that is, as WROWS–MAXWIN–1). When a new MINWIN value is selected (by %D or %B), it alters the secondary input value if secondary input is selected at the time, rather than the main file MINWIN value. Whenever an increased minimum value is selected for secondary input, this has the effect of constraining the maximum size of the main file's effective window, and conversely. If no space at all is left for secondary input, a default size is forced as necessary.

Where there are no performance restrictions imposed by communication or processing systems, the upper limit value (MAXWIN) is usually appropriate for MINWIN. The default value is chosen according to the system.

In hard-copy mode, MINWIN is used to control the number of lines to be printed as feedback after command execution. A value of zero suppresses feedback.

WIDTH – text line width Default: 80 Range: 5:256

The parameter WIDTH specifies the maximum line length to be used in connection with the Adjust and 'At' commands (and for failure conditions in the case of Insert and Join). Note that it applies only in these contexts, and does not imply any general restriction on line length. The customary initial default value is 80.

MARGIN – left margin Default: 0 Range: 0:WIDTH–1

The parameter MARGIN specifies a left margin position, which defines the effective start of the line. This determines where the file pointer is placed following a Move or Move Back command, and is also relevant to the operation of the Adjust command.

MATCH/NOMATCH – ignore/heed case Default: MATCH

When matching text strings in the course of executing any of the commands Find, Delete, Traverse, Uncover or Verify, the Editor may or may not ignore case distinctions between letters. By default it ignores them. This mode of matching may be switched off by selecting the option "–NOMATCH" and re-established by selecting the alternative option "–MATCH". When it is switched off, letters in

text parameters are matched exactly as typed against letters in the file.
The setting of this mode does not affect text parameters for the insertion commands, which are always inserted exactly as typed.

HILIGHT/MARK – show file pointer by highlight/marker

This parameter controls how the current position of the file pointer is displayed when the Editor is in command mode. HILIGHT implies use of whatever capability a video has for distinguishing arbitrary characters, for example, reduced or increased intensity, underline, or (preferably) reverse video. Its use may require particular switch settings or intensity adjustments on the terminal.
MARK implies the technique of overwriting the character immediately to the left of the current position with a distinguished character (splodge or tilde). An additional column is inserted at the beginning of each line in this mode, for use when the pointer is at the beginning of the line. MARK is appropriate for terminals which have no means of highlighting individual character positions.

EARLY/LATE – update window on reaching/passing bottom line

The default strategy for updating the window is to do so only when the material in it is changed or the file pointer is moved outside it. Selecting EARLY causes the window to be extended or refreshed when the bottom line is reached rather than when it is passed.

%E Environment command

The Environment command switches the Editor into a mode in which the various options described above may be modified. It cycles through the list of modifiable options displaying the current value and permitting an alternative value to be specifed. The RETURN key is used to move on to the next in the list and colon is used to return to editing. For a numeric option, a number must be typed to specify a new value. For an on/off option any response other than RETURN or colon is sufficient to alter the setting.

%D*n* Display size *n*
%D Re-display

This command provides an alternative means of setting MINWIN, without entering Environment setting mode. When used by itself (without *n*), it leaves MINWIN un-altered but re-writes the display. It should be used if for any reason the window has been corrupted (by an operator message, for example).

%L*n* Line width *n*
%M*n* Margin *n*

These commands provide an alternative means of setting WIDTH and MARGIN.

%S *file-name* Secondary input definition

As an alternative to specifying a secondary input file at the time of calling the Editor, a command of the form %S followed by a file-name may be given during the course of editing. This establishes the named file as the secondary input and switches to it. Any existing secondary file is discarded. In some implementations, the amount of information which can be added from a file specified in this way may be limited, compared with what can be inserted when the secondary input file is specified at the outset.

%G *file-name* Get commands from file

The parameter PRE described under Options permits an initial set of commands to be read from a file. The %G command allows a command file to be nominated at any time during the course of editing, for example, to allow a different set of macros to be set up, or to invoke a complex but stereotyped sequence of editing operations.

%P *file-name* Put key definitions to file

This command allows all the key definitions which have been made since the start of the current editing session to be saved in a specified file for subsequent recall (as PRE or via %G). Definitions of control keys are represented in a coded form, using only printing characters.

%W*n* Wipe record of *n* deleted lines

The fact that the Editor retains a record of all lines deleted from the file, against the possibility of later re-insertion, can sometimes be a nuisance, since lines that are definitely not wanted again may get in the road of those that are. The Wipe command %W causes the record of the last *n* deleted lines to be lost so that they become irrevocable.

It may also be necessary to use Wipe in order to release space to bring in a large amount of data from a secondary input file to replace a large chunk deleted from the main file.

6. COMMAND AND TEXT MACROS

The upper-case letters A to W and the punctuation symbols have a fixed significance to ECCE. All the lower-case letters are available for definition as command macros. The upper-case letters X to Z are available for definition as text macros. Control keys can be defined as command or text macros. Any of these keys can be defined using the (%K) (Key definition) command.

The printing keys mentioned have no macro significance in data-entry mode or within text strings, where they stand for themselves. In commands, any of the lower-case letters may be used in a position where a command letter is expected and may stand for any sequence or partial sequence of commands, while any of the letters X to Z may be used in a position where a quoted text string is expected and may stand for any string of characters.

As well as being definable by %K, the letters X to Z and x to z can be defined to stand for a text sequence appearing in the file being edited, or the secondary input file, using the Define Macro (~) command.

The control keys are always interpreted as macros, in one of two ways, depending on how they have been defined. One form of definition forces interpretation of the sequence of characters for which the key stands as a command sequence; the other causes the use of the key to be equivalent to typing the characters explicitly as part of the text being entered. The distinction is indicated when defining one of these keys by using a colon to define a command sequence and an equals-sign to define a direct replacement sequence.

Initially the lower-case letters a to w are defined to be equivalent to their upper-case equivalents. Where command letters appear within a macro definition, it is sensible to use the upper-case form if it is intended to utilise the basic meaning of that command letter, lest the lower-case form should have been re-defined.

Note that the content of a command macro is interpreted, not when the definition is made, but when the macro is used. When an exclamation-mark is used in place of a text parameter within a sequence invoked as a command macro, the effect is to cause the actual text parameter to be sought at the point following the macro call, rather than from the command stream. This allows a limited degree of macro parameterisation. Here again any of the forms of text parameter - quoted string, text macro letter, ditto, or exclamation mark - are valid.

%Q Key enquiry command

The Special command %Q is provided as a way of finding out the current significance of any key. It may be used as a one-shot command or to cause entry to Enquiry mode. One-shot use involves pressing the key about which information is required immediately after the %Q (followed by RETURN if not a control key). Entry to Enquiry mode is indicated by typing just %Q followed by RETURN. In this case the Editor continues to prompt for keys to be explained until a colon is typed; as before, a control key does not require a following RETURN.

For any basic editing command letter, a brief indication of the meaning of the letter is given. For a key defined as a macro, the current definition of that macro is printed out; for a multi-line sequence, only the first line is printed out.

%K Key definition command

The Special command %K is used to define or re-define keys. Like %Q, it may be used as a one-shot command or to enter a Key-definition mode. In the first case, a single definition is entered along with the %K, while in the definition mode, the Editor continues to prompt for definitions until a leading colon is typed.

Each definition takes the form of the key to be defined followed by a colon or equals sign followed by the text making up the definition. When a control key is being defined, depression of the control key causes an asterisk to be echoed. It is important to choose the right separator for the effect wanted. The use of colon indicates that the key is being defined as a command execution sequence. Use of equals as separator indicates that the key is being defined to stand for a plain text string which will be directly echoed when the key is pressed.

To cover the case where it is realised after typing in an ordinary command that it could usefully have been defined as a macro, an alternative form of definition is provided to define the key to be the text of the last explicitly typed command line. This is: key to be defined followed by the ditto symbol ("). For example the pair of commands:

```
F/1984/ S/1985/
%K x"
```

would capture the Find and Substitute command as the definition of x.

In order to define a control key as the mode switch key, it should be defined as standing for the 'command' backslash. That is, after typing %K and the key, type colon, backslash, RETURN.

Use of command macros

After the definitions

```
%K x=F/basically/
%K y=MR*1/  :
```

the following equivalences would apply

```
xS/actually/-      F/basically/S/actually/
M-10 x T/y/-1/./   M-10 F/basically/ T/y/ 1/./
ycomment/ -        MR*1/  :comment/
```

A command macro letter may abbreviate any initial part of a command sequence; the last example illustrates a case where it includes the opening delimiter for a text string and a fixed initial part. Where a macro letter is defined to be a complete group of two or more commands it is always sensible to include parentheses in the definition, as in

```
%K z=(F/error/1/**/)
```

rather than just%K z=F/error/1/**/
so that if a repetition count is attached to the macro letter, it will apply to the whole sequence and not just the last component (see next chapter).

The definition of one command macro may include a reference to another, but any form of circular definition is invalid and an occurrence of any of the macro letters within text delimiters always stands for itself.

One common case where temporary macros can be useful is when some but not all occurrences of a text string, say max, have to be changed to something else, say count, inspection being required to determine which. With the two definitions

```
%K x=F/max/
%K y=S/count/x
```

x can be used to find the first occurrence of max and thereafter y or x depending on whether a change is required or not.

Macro definitions persevere until the end of the editing session or until the relevant key is re-defined.

7. PROGRAMMED COMMANDS

This chapter describes facilities for constructing more powerful commands from simple commands. So far, the only form of compound command structure introduced has been the facility to type more than one command in a single command line, with the consequential capability of repeating the complete sequence by subsequently typing a repetition number instead of another command.

The structuring facilities described in the following sections can be used to carry out some quite complicated operations, but commands making heavy use of them tend to become rather difficult to understand. It is not sensible to try to devise a programmed command of any significant complexity while working at the terminal. Contemplation in tranquillity is required. Some users note in a log-book those which they have found useful in the past.

Command sequences

Any sequence of commands may be enclosed in parentheses and treated as a single command. In particular this permits a repetition count to be attached to a sequence of commands. For example, the command

```
(F.integer.I.%.)3
```

has the effect of inserting a percent sign in front of the next three occurrences of "integer". Compare

```
F.integer.I.%.3
```

which inserts three percent signs in front of the first occurrence of "integer" only, and

```
F.integer.3 I.%.3
```

which inserts three percent signs in front of the third occurrence only. For this simple case, it would be almost as convenient to type the pair of commands as one command line, and then type a 2 to repeat them twice more, but that option would not cover cases where the bracketed sequence is only part of a complete command line.

A command sequence fails when any component of it fails, so that an asterisk attached to a bracketed command sequence specifies indefinite repetition until one of the contained commands fails. As a check against infinite looping, a limit of 10,000 iterations is applied to the repetition of bracketed commands.

Alternative command sequences

Another form of compound command is one providing a number of alternatives. This consists of two or more individual commands or command sequences separated by commas. Execution starts with the first command and if that alternative is completed without failure, the other alternatives are ignored. If any of the commands making up the first alternative fails, then the second alternative (following the first comma) is executed, and so forth. Only if a failure occurs on the last alternative is the whole compound command considered to have failed. For example, it might be required to include, as a component of à more complex command, an instruction to position the pointer at the next space on the line, or at the end of the line if there is no space to the right of the pointer. A possible command to achieve this would be

```
(F1/ /. R*)
```

that is, try to find a space on the current line, which failing move to the right of the line.

Consider also the problem of interchanging two text strings, say "basically" and "actually", throughout a complete file. Obviously, the commands

```
(F/basically/S/actually/)*
M-*
(F/actually/S/basically/)*
```

would end up with all occurrences of both words converted to "basically". One solution would be to convert all occurrences of the word "basically" to some unique sequence of characters, make the other change through the file, and finally convert the unique character sequences to "actually". A preferable approach, using alternative sequences, permits this kind of edit to be made progressively on a single pass through the file. First consider the simple sequence

```
(R,M)
```

Right-shift fails only when the pointer is at the end of a line, so that this sequence performs a Right-shift except at the end of a line, when it performs a Move. The Move, and hence the compound command, fails only at the end of the file. Accordingly this is a command which makes it possible to 'inch' through a file on a character by character basis. The case under consideration can be handled by expanding this sequence to include alternatives to test for the two text strings and make the necessary change, leading to:

```
(V/basically/S/actually/, V/actually/S/basically/, R, M)*
```

This framework, using a set of Verify commands together with the 'inching' sequence "R,M", is one that can be used for a variety of different requirements.

Inverted failure condition

If a command, simple or bracketed, is followed by the symbol \, the failure condition for the command is inverted so that successful execution causes a failure and unsuccessful execution does not. This in no way alters what the Editor attempts to do by way of carrying out the command, which has its customary effect, if any, except that it is deemed to have failed when it has not, and conversely. This curious effect is sometimes useful, most obviously with the Verify command. For example, V/+/ makes it possible (in effect) to make the following command conditional on there being a plus-sign immediately to the right of the pointer. Accordingly, V/+/\ makes the following command conditional on there not being a plus-sign immediately to the right of the pointer.

Cancelled failure condition

If a command, simple or bracketed, is followed by the symbol '?', any failure condition arising in the execution of the command is cancelled, that is, any further action is taken on the basis that the command succeeded. As with inversion, this in no way alters the effect of the command itself. For example, suppose it was required to insert an ampersand in front of each line in a file which started with a exclamation-mark. The sequence

```
(V/!/ I/&/ M)*
```

is not adequate since the part within brackets will fail on any line not starting with an exclamation mark and cause termination of the whole sequence. It is only the failure of the Move which should cause failure of the sequence. Bracketing the Verify and the Insert and appending a question-mark achieves the desired effect:

```
( (V/!/ I/&/)? M)*
```

Further examples

The examples which follow either might be useful in themselves or illustrate general techniques. Many such commands are not rigorous, but depend on the originator of the document maintaining a consistency of style in terms of such matters as the inclusion or omission of spaces.

```
(a)  (MR)*                     Find first blank line
     (MR\)* L                  Find first non-blank line
```

The command Right-shift fails at the end of the line. Immediately after a Move, it will fail only if the line is blank. Hence (MR*) causes Moves to be executed until either the end of the file or a blank line is reached. Inverting the failure condition on R locates the first non-blank line, but note that a final L is required if the pointer is to end up at the start of the line.

```
(b)  (RI/ /)* E-               Double-space a line
```

This sequence inserts a space to the right of each existing character (including spaces) on the line. The final E- removes the last space inserted, because trailing spaces can be a source of confusion.

```
(c)  (R* (L D/ /)* M)*         Delete trailing spaces
```

This command will eliminate any trailing spaces that may have crept into the file. On each line, the pointer is moved to the end and then successive attempts are made to Left-shift and Delete a space.

```
(d)  ( (RLI/ /4)? M)*          Create left margin of 4 spaces
```

The obvious command would be simply (I/ /4M)*, but that would add spaces to blank lines, which is undesirable. Hence the RL to check that the line is not blank.

```
(e)  F/sin/(V/sin(/,S/evil/)   Replace selected occurrences
```

The form of this command illustrates the case where it is required to pick out certain occurrences of a word for alteration but not others. A programmer has inadvertently employed sin as a variable in a program which also makes use of the mathematical function sin(...). The command sequence locates an occurrence of sin, then verifies that it is followed by a left parenthesis using V/sin(/, or else changes it to evil.

```
(f)  ~ M ~ IX                  Replicate line
```

This is an example of a general technique for replicating material in the file, by using ~ before and after the text in order to capture it as the definition of X.

8. COMMAND CHECKLISTS

A	*21*	Adjust line length
B	*18*	Break line in two at pointer position
C	*17*	Case-change with right-shift
C-	*17*	Case-change with left-shift
D/*text*/	*17*	Delete first occurrence of *text*
D-/*text*/		Delete prior occurrence of *text*
E	*17*	Erase character to right of pointer
E-	*17*	Erase character to left of pointer
F/*text*/	*14*	Find first/next occurrence of *text*
F-/*text*/	*14*	Find prior occurrence of *text*
G/*text*/	*16*	Get (insert) *text* as complete line above current line
G-	*19*	Get back deleted line
I/*text*/	*15*	Insert *text* to left of pointer
I-	*19*	Insert back deleted character
J	*18*	Join next line to current
K	*16*	Kill (delete) current line
K-	*16*	Kill (delete) previous line
L	*13*	Left-shift one character position
<	*13*	Cursor Left
M	*12*	Move forward one line
↓	*12*	Cursor down
M-	*12*	Move Back one line
↑	*13*	Cursor up
#*n*	*12*	Move to line number *n*
N	*20*	Next - locate next word / bracket
N-	*20*	Next Back - locate previous word / bracket
O/*text*/	*15*	Overwrite existing text with *text*
O-	*19*	Overwrite back character (undo)
P	*19*	Print line on terminal
Q	*20*	Query Spelling
R	*13*	Right-shift one character position
>	*13*	Cursor Right
S/*text*/	*16*	Substitute *text* for text last found
T/*text*/	*14*	Traverse first occurrence of *text*
U/*text*/	*18*	Uncover first/next occurrence of *text*
V/*text*/	*15*	Verify presence of *text* at pointer position

@*n*	*21*	Align to column *n*
+*n*	*20*	Increment number by *n*
$	*23*	Switch inputs
^	*22*	Set Marker
^1,...,^6)	*22*	Define Macro letter X,Y,Z,x,y,z
=	*22*	Revert to Marker
|	*22*	Toggle Destructive mode

Special commands

%A		Abandon edit without updating file
%C	*9*	Close edit normally
%D	*27*	re-write Display
%D*n*	*27*	set minimum Display (MINWIN) to *n* and re-write
%E	*26*	alter Environment options
%G *file*	*27*	Get commands from file
%H	*9*	obtain Help information
%K	*29*	Key definition
%L*n*	*27*	set Line width (WIDTH) to *n*
%M*n*	*27*	set left Margin (MARGIN) to *n*
%P *file*	*27*	Put key definitions to file
%Q	*28*	Query key definitions
%S *file*	*27*	define Secondary input file
%W*n*	*27*	Wipe out record of *n* deleted lines

Reserved symbols

( )	*31*	command grouping parentheses
,	*31*	separator for alternatives
\	*32*	suffix to invert failure condition
?	*32*	suffix to cancel failure condition
"		ditto text indicator
! initial		prefix for system command
non-initial		direct-entry text indicator
%		prefix for Special command

V I S U A L   2 0 0   C O N T R O L   K E Y S

cursor control keys -- standard pointer movement

RETURN	: M Move to start of next line
HOME	: H Move to top,left,right,bottom depending on last cursor movement
BS	: G- Recover deleted line
LF	: 1 repeat last typed command
TAB	: N Move to Next word (or close bracket)
SHIFT + TAB	: N- Move to previous word (or open bracket)

	7 IL	8 IC	9 ST
	G* Insert lines until next control	I Insert text	I. . Insert space
CF	B Break Line	I-* Recover chars	O-* Undo
	4 DL	5 DC	6 CT
	K Delete line	E Erase char	D. . Delete space
CF	G- Recover line	I- Recover char	IX Insert X
	1 EL	2 EF	3 EP
	E* Erase rest line	S! Substitute text	^ Set Marker
CF	J Join next line	S" Substitute same	$ Switch Inputs
	0	, CPY	. PRT
	F! Find text	%C Close edit	%D Re-display
CF	F" Find same		I Toggle destructive mode
	-	ENTER	
	M-* Move back to top of file	\ Switch between command and data entry mode	

The entries on the lines marked CF require simultaneous use of the CONVERT FUNCTION key

The function keys are also eligible for definition on those terminals which have them

ERCC ENUMERATION OF VIDEOS

code	terminal	highlight	part-screen scroll
0	unspecified	-	-
1	hardcopy width 72	-	-
2	hardcopy width 80	-	-
3	hardcopy width 132	-	-
4	unknown video	-	-
5	ITT	-	-
6	Perkin-Elmer Bantam	-	-
7	Lynwood	-	-
8	DEC VT52	yes	-
9	micro	-	-
10	ADM-3A	-	-
11	Visual 200	yes	yes
12	VT100	yes	yes
13	Hazeltine Esprit	yes	yes
14	Hazeltine 1500	-	-
15	Newbury 8000	yes	yes
16	Pericom	-	-
17	Tektronix 4010	-	-
18	IBM 3101	-	-
19	Dacoll 242E	-	-
20	Volker Craig 404	-	-
21	ICL KDS7362	yes	yes
22	Hazeltine Esprit II	yes	yes
23	Hazeltine Esprit III	yes	yes
24	ADM-5	-	-
25	Visual 50 & 55	yes	yes
26	Tektronix 4014	yes	-
27	Datatype X5A	yes	yes
28	ANSI compatible	yes	yes