THE EDITOR
SPECIAL EDITION

by
Charles Dillon

DIGITAL PRECISION

Published by Freddy Vachha, 222 The Avenue, London E4 9SE

1. Purpose and applicability

1.1 What does Editor do

The Editor is designed to inspect, create and amend 'text' files. Many features are provided to enable the rapid creation and alteration of text, including a full set of cursor movement, screen paging, windowing and scrolling commands; simple and extended "find" and "exchange"; "block" definition and manipulation commands; file reading, writing, merging and fragmenting.

The program is distinguishable from a dedicated 'word processor' in several ways. The primary distinction is that Editor is capable of operating on all types of files - not exclusively "document" files. Almost all commonly available word-processing features are provided within Editor, most usually in a form that is equally applicable whatever the file type. In addition, Editor provides many features not to be found in standard word processors, that enable powerful and flexible manipulation of text.

The most distinctive feature of the Editor is its "lazy screen" philosophy. The program is constantly ready to respond to user input - text, commands or whatever - even if the program is part way through doing something else (which will typically be updating the screen). This has the effect of greatly increasing the effective response of the program and thereby improving actual work throughput. The next significant feature of the program is the very extensive and powerful command set, which when used fully, can take on some aspects of a programming language.

1.2 Who will find it useful

The Editor will be recognised as a very powerful tool by anyone involved in any type of program development, especially high level languages such as C, Basic, Pascal, Archive, etc. The program provides a convenient and rapid method of generating libraries of modules, inclusion of modules, implementing global changes, checking redundancy, re-shaping etc etc., as well as for generating original source code.

Equally, the Editor may be used for minor amendment or massive reformatting of any other type of text file - for example directories of volumes (disc/tape), export files from Archive, Abacus, Easel, print files from Quill etc etc.

A further application, using the powerful extended command repertoire, allows the rapid generation of test data files.

Not least, the program may be used for the fast creation of lightly or heavily formatted documents (this User Guide for example).

Some users will find that the Editor is also quite suitable as a database manager, allowing lists (names and addresses, or parts, or catalogue etc) to be created, maintained and examined, with a considerable amount of ease, flexibility and power.

1.3 What types of file

The program is most useful when operating on 'text' files. These are files that typically contain only 'displayable' characters, orientated as 'lines' - i.e. terminated by a line feed control character.

In actual fact, the program will read any file it is asked to (even if it is not a text file), and do the best that it can to make a meaningful display of the file contents. Editor may be used to inspect or make simple or complex changes to object programs, screen dumps, databases, dedicated document files and so forth.

Very few restrictions are placed on the operations performed on a file, irrespective of its file type. The user may therefore do what s/he chooses, in whatever order that s/he chooses.

Section 8.15 contains more detail about the handling of files.

No restrictions are placed on the data that is entered by the user - normal text is obviously catered for, but so also are control characters (ASCII 0 to 31) and those characters in the ASCII range above 127. However some characters with codes above 127 must be entered in a special way - via the command line.

Section 4.3 contains more detail about the entry of data.

The program is supplied with a special character font, so that normally non-display (control) characters are given a unique representation on screen. The font specification is supplied in standard QL format, and may be replaced, if desired, by any other properly generated font.

If we have anything to say over and above what is said in the manual, we will put it in a file called UPDATES which will be on the Special Editor disk/mcs. To read this file load it into Editor using the RU command.

1.4 Using this guide

This guide is intended both as an introduction and as a reference document.

The scope and flexibility of the program are such that a wide variety of users will find benefit from using Editor for a whole range of different tasks.

This guide attempts therefore to describe in detail all of the features of the program. Some of the facilities of Editor may be invaluable to some people and irrelevant to others. The precise mix of commands and features used is likely to vary considerably from one individual to another.

The intention has been, wherever possible, to separate descriptive sections and reference material.

Sections 1 to 5 are a general introduction, and really should be read to get an overall understanding of the program.

Sections 6 and 7 introduce the commands. Immediate commands are fully described in section 6 - the user new to Editor will find few surprises here, since almost all of the commands will be familiar from normal operation of the QL. Section 7 is a description of the use of Extended commands - how to enter and edit them and so forth.

Sections 8 and 9 are reference sections, detailing all Extended commands and all Error messages respectively. It is not the kind of thing that you are likely to want to sit and read all in one session, but section 8 particularly is a mine of information.

Section 10 works through a few examples - "here is a job; how do we use Editor to get it done".

Section 11 describes the features and operation of the special Configuration program supplied with Editor. This section should be read, and the program used, so that you may set up Editor to operate in exactly the way you require it.

Sections 12 and 13 provide information on transferring files between Quill and Editor, and on getting Editor files printed, using among other techniques the supplied print program "edtprt_bin".

The documentation on the Editor is fairly extensive. We have provided so much detail because we want the purchaser to get as much benefit as possible from the program. It is of course easy to use the program very effectively WITHOUT an intensive course of study of this text.

Some material in the text has been "boxed". On a first reading of this guide, it is not necessary to read these text boxes. The subject of the box will generally be some obscure or low frequency use or facility.

2. Machine requirements

2.1 QL hardware

This version - version 2 - of Editor requires a memory expansion to be fitted to the QL. The program may well load on a standard QL, but no useful work may be done in Editor on a small machine. An An earlier version - version 1.17 - is the last planned version capable of operating on an unexpanded machine. The earlier version remains available from Digital Precision. The facilities provided by the smaller edition are a subset of those described here.

The display device may be either a monitor or a TV, but as usual a monitor is preferable. Whatever devices are present on the system (discs, hard discs, ram disc, tapes, serial or parallel ports, network ports) will be properly used by Editor, if so directed.

When the program is asked to operate on a file, the whole of the file is read into memory. Clearly, the amount of memory available on the QL determines the maximum size of file that the program can handle.

2.2 Superbasic extensions

The Editor is written entirely in Superbasic, compiled using the TURBO compiler from Digital Precision.

Certain commands used in the program are not normally part of the Superbasic language. These additional commands are provided in the form of an extensions file - widely known as the TURBO Toolkit. The extensions file must be loaded and linked into the system before Editor - or its Configuration program - may be run. This is achieved within the supplied "boot" procedure. The loading and linking does not need to be done each time the program is run, but must be done after switching on the QL or after a system "reset".

Please note that if you already have and use the TURBO Toolkit, then you should use your extensions file in preference to the supplied extensions file, which is the RUNTIME version of TURBO Toolkit. For best results and maximum compatibility with other programs, you should use Toolkit version 2.0 or later.

2.3 Supplied Files

The program is (normally) provided on a disc or two microdrive cartridges. The supplied media contain several files. The name and purpose of each file is as follows: .

boot	Startup file optionally used to start Editor
edt_bin	The Editor program file
edt_charset	The special character set
edt_help	The on-line help file
edt_config_bin	Program to tailor Editor to your requirements
xtras	The extensions file
boot_cmd	Sample start-up command file
demo_cmd	Demonstration command file
column_cmd	Demonstartion command file
quill_cmd	Demonstration command file
updates	Any 'extra' comments
edtprt_bin	File printing program
driver_dat	Configuration data for your printer

As with any software, you are strongly advised to treat the supplied media as MASTER copies - specifically not for use.

NOT EVEN ONCE.

You should take a copy of all of the supplied files, before even the most cursory of test sessions. Most memory expansions for the QL have some kind of toolkit that will enable generic copies to be taken quite simply; WCOPY in the QJUMP Toolkit will do the job.

The supplied media should then be stored away in a safe place.

3. Starting the Editor

3.1 Setting up the system

When the Editor is to be started, the QL will be in one of two states - either the required extensions have been loaded or they have not.

If the extensions are in the machine, then the Editor may be started simply by the command "exec DVC_edt_bin", where DVC is the name of the device which holds the program (e.g. mdv1 or flp2 etc). If, on loading, the program senses that the required extensions are not in fact resident, a series of messages is displayed and the program terminates.

If the extensions are not resident, then they should be loaded before invoking the Editor. This may be accomplished by the command "lrun DVC_boot" where DVC is the name of the device which holds the boot file.

Once the program has been successfully loaded, the identification screen is displayed. On this screen is shown the version number, copyright details and an invitation to invoke the Help information, if required.

It is not fatal but quite inefficient of memory space to load the extensions several times. For example, if the extensions have been loaded and a second copy of Editor is required in the machine, then DO NOT use 'lrun DVC_boot' to start the second or subsequent Editor, but simply use the 'exec' command, or equivalent.

Additionally, if the Configurator program is to be run, the same extensions must be in the machine. If the extensions are not loaded, probably the easiest way to load them is to proceed as though you were starting Editor - i.e. the same LRUN command - and then quit from Editor when it has started.

3.2 Help file

Initial Help information is displayed if the F1 key is pressed in response to the identification screen - or at any subsequent time during the operation of the program.

The Help text is organised in several pages. The first page gives gives information on the "immediate" commands - cursor movement, screen paging, scrolling etc - and the following pages summarise most of (if not precisely all of) the "extended" commands. The final page contains some reminders on the use of command files.

The amount of information to be contained in these pages is considerable. When displaying the Help text, the program sets the screen mode to "high resolution", irrespective of the current screen mode selected by the user.

- Help information can only be displayed if the Help text file is available on the 'nominated' device. Editor has to know which device to search for the Help file. This is specified when 'personalising' the Editor using the Configurator program - see section 11. If the Help text file is not found, a message is displayed.

The Help file supplied is a standard text file, and can of course itself be edited within Editor. Some users may wish to examine and or amend the contents of this file to highlight the particular commands or groups they choose.

Each item (line) in the Help file is constructed to a similar format, being page number, line, column, ink colour, paper colour, continuation indicator and line text. It is strongly recommended that if amendments are to be made to the Help file, a copy of the original should be kept in a safe place.

The Configurator program allows the name of the Help file to be changed. If the standard file is modified, it is a good precaution also to change the name of the file.

3.3 Character set file

During the start-up seqrence, Editor automatically searches for the 'character set' file om the default device. If this file is found, it is loaded into memory. It is the contents of this file that allow the otherwise nom-display characters (ASCII values 0 to 31) to be displayed in a visually distinguishable manner.

It may be that the facilities provided by the special character font are not required for certain files, or it may be that memory space is at a premium.

Under either of these circumstances, the user has the option of not making the character set file available at load time - e.g. change the file name, or delete the file from the system (working) volume, or simply to remove the volume from the device on program start-up.

This will have the effect of allowing another 1200 bytes of work space (approx) for the accomodation of edit files.

The character set (font) file supplied and used by Editor is in standard QL format. If it is required that another font file should be used, then two 'problems' will need to be solved. The first problem is the creation of a suitable font file. This can be done with any of the proprietary font designers available for the QL - for example, EYE-Q or TURBO Toolkit from Digital Precision.

The second problem is to get Editor to access the new font file. This can be done either by calling the new file "edt_charset" or by using the Configurator program to alter the name that Editor expects the character set file to have.

3.4 Start-up memory

When Editor starts up, it allocates a small amount of work space from free memory on the system (about 15K). This area is used to contain the text and control information of the file to be created.

If an existing file is to be read, Editor will release the current work space, and request from the system an amount of memory consistent with the size and type of the file to be read.

Under normal circumstances therefore, there is no need to be concerned about memory allocation. For further comments on memory considerations see sections 8.11 and 8.15.

3.5 Multi-tasking managers

Some multi-tasking programs available for the QL allow the user to specify the maximum amount of memory to be allocated to each of the tasks to be initiated. This facility is actually a method of controlling some vintage programs (e.g. the Psion suite) - it stops them from annexing all of the system memory. Editor's memory management is "well behaved" and no memory limit need be specified.

Certain technical aspects of the manner in which TURBO works (TURBO is used to compile Editor) have the consequence that some features of QRAM and TaskMaster should not be used with Editor.

Specifically, for TaskMaster, when Editor is initiated either from the configuration selection or from the 'start new program' option, the "shared code" feature MUST NOT BE USED.

For identical reasons, Editor may be started and operated under QRAM - either with EXEC from SuperBasic or from the "Files" menu of QRAM -- but no attempt should be made to set up Editor as a "hotkeyed" program.

To streamline screen effects under QRAM, a new screen channel is opened at the start of program. This channel is maintained at the largest screen area required - other than the help screen. If the working screen is re-sized and/or repositioned, the guardian window is appropriately modified. This avoids strange screen effects previously noticeable under QRAM when the command window and others were opened in Editor.

To the best of our knowledge at this time, no other restrictions exist. Editor and TURBO Toolkit adhere to ALL of the rules for applications programs on the QL. Editor will therefore operate faultlessly with any other program(s) that also obey the rules.

A few notes on Multi-tasking:

The QL is a multi-tasking machine. This means that it is capable of running more than one job at a time. The QDOS operating system maintains lots of lists and queues to distinguish resources being used by one program from those being used by another. QDOS is itself a competent multi-tasking manager.

Much of the early software for the QL was unaware of - or at least took no advantage of - this facility. For example, the Psion programs each expect to be the only program in the machine.

If several programs (tasks, jobs - these words are often used interchangeably to mean the same thing) are operating in the machine at one time, QDOS acts as an arbitrator and resource allocator. For the present, the resource of most interest is the keyboard. If several jobs are waiting on keyboard input, only one job may "own" the keyboard at any one time. If the job using the keyboard terminates, QDOS reallocates the keyboard to the 'next' job waiting. (This will usually be SuperBASIC, since SuperBASIC is always in the machine and is almost always looking for input).

QDOS supports a feature that allows the system user to request the reallocation of the keyboard. The convention is to press the key combination CTRL and C (holding the CTRL key down, press the C key). What happens is the cursor which is flashing on the screen stops flashing - the job that had the keyboard is now suspended. QDOS allocates the keyboard to the next job in the queue headed "waiting for keyboard", and its cursor will start to flash - meaning that keys pressed on the keyboard will now be seen by the second program rather than the first. This feature operates on a round robin basis, so by successive use of CTRL/C, you end up back at the original program, however many tasks there may be operating.

Since Editor is written entirely to "QDOS standards", the above describes precisely how the user may jump into or from Editor at any time that the program is running. Equally, assuming there is sufficient RAM available, several copies of Editor may be present in the machine at the same time - presumably (but not necessarily) working on different files. By setting up the screen size, position and colours distinguishably, the user may freely transfer from one Editor copy to another without confusion. These screen attributes may be set either by the Configurator program, or dynamically while Editor is running.

3.6 Start-up command file

As an additional convenience, immediately on starting, Editor searches for a start-up file, the default name for which is "boot_cmd".

If this file is found, on the 'system' device - the same device that carries the character set and help files - then it is executed as a standard Editor command file. This allows a standard environment to be specified once and for all, or changed at will.

4. Using the Editor

4.1 Screen layout

Once the initial sequence has been completed, the user is presented with the 'working' window. The default start-up window for the EDITOR has been made horrible to encourage you to use the Configurator.... This program allows you to change many aspects of the EDITOR's operation - file defaults, device defaults, screen size/placement/colour etc etc, so do look at it. The facilities and operation of the program are described in section 11.

The main (upper) screen area is for the display and entry of the text. The small area at the bottom of the screen - the last line - serves a triple function. Most often, the line displays status information - indicating the current line and column position of the cursor, the number of lines in the file, and the data entry 'mode'. The 'mode' is either "overstrike" mode or "insert" mode.

When entering or executing extended commands, the status information is displaced, and the bottom line is used as the command line interface.

The third use for the bottom line is for the display of error or warning messages when an exception condition occurs.

The main screen, command/status line and error messages each have individually specifiable ink and paper colours, the intention being to differentiate them from each other.

4.2 Screen and keyboard handling

There are four instances of keyboard use:

Entering text
Issuing an immediate command
Issuing one or more extended commands
Responding to a program prompt

Whenever the Editor receives data from the keyboard, it gives absolute priority to dealing with the data input, if necessary suspending any other current activity, except when reading or writing a file.

When text is being entered, the program is principally occupied with servicing the data being entered, and maintaining the current screen line in correspondence with the keys depressed. Most of the other presumed responsibilities of the program - maintaining the status line, and keeping the rest of the screen updated etc - are given second priority. When a pause in data entry is detected, the program catches up with the "housekeeping".

This effect will be most noticeable when a data line exceeds the nominal screen width - the whole screen should really be "panned" sideways, since the current line is being panned sideways. If the program finds that it has the time to do this then it will, but if the data entry rate is high, then it may decide to leave the rest of the screen where it is and sort it out a little later.

If an immediate command (see 4.4) is entered, the same sort of action takes place.

If an extended command is invoked (see 4.5), the current activity on the main screen (if any) is suspended and the command window is opened at the bottom of the screen, ready for the entry of the commands.

Prompts from Editor will always require a "yes/no" type of response - e.g. to overwrite a file, or to confirm a text substitution etc. In this latter case, even if main screen activity has been suspended, Editor will display the line on which the text substitution is to be confirmed.

4.3. Data entry and the Status line

The Editor is designed primarily for the processing of text files. Normal text characters are entered as usual at the position shown by the cursor. All characters in the ASCII range 0 to 127 - the standard set - may be entered from the keyboard.

Special provision has been made for the 'control' characters 0 to 31. The range 1 to 26 may be entered using the sequence CTRL a to CTRL z. There are four awkward characters - CTRL space, CTRL c, CTRL i and CTRL j. These are awkward because they get trapped by the keyboard or screen handlers in QDOS, the operating system.

CTRL £	may be used instead of CTRL space	to get '0'	(NUL)
CTRL/SHIFT/c	may be used instead of CTRL c	to get '3'	(ETX)
CTRL/SHIFT/i	may be used instead of CTRL i	to get '9'	(TAB)
ENTER	may be used instead of CTRL j	to get '10'	(LF)

The range 27 to 31 may be entered as follows:

ESC	gets '27'
CTRL/SHIFT/\	gets '28'
CTRL/SHIFT/]	gets '29'
CTRL/SHIFT/£	gets '30'
CTRL/SHIFT/ESC	gets '31'

Each control character has a unique representation on screen. The first 31 characters have a 'high bar' above them. Character 0 is shown as a double bar (a raised = sign), characters 1 through 26 are shown as the capital letter A through Z with the bar above. The penultimate four are shown as up arrow, down arrow, left arrow and right arrow, each of course with the bar above. The final character (31) is displayed as a triangle. This character serves a special purpose ("non-break space") with 'document' files, as described later (see sections 5.1 and 8.8).

The Status line is updated as each character is entered. The format of the Status line is determined by the chosen size (width) of screen. If the screen is wide enough, the status line displays:

```
Line: lllll  Col: cccc  Line count: ttttt  Mode: xxxxxxxx
```

For a narrower screen, the display is:

```
Line/col: lllll/cccc ttttt x
```

For the narrowest screen size, the display becomes:

```
lllll-cccc ttttt x
```

In each of the above, the letters lllll are the current line number, the letters cccc are the current column, the letters ttttt are the total number of lines in the file. The string xxxxxxxx reads either "O/Strike" or "Insert" depending on the data entry mode. In the other cases the character x is either "O" or "I".

If an 'unformatted' file is being edited (see section 8.15.2), the status line also displays the ASCII code of the character under the cursor - i.e. the character that the cursor is sitting on. If the cursor position is beyond the logical end of the line, no code at all is displayed.

If a 'document' file is being edited (see section 8.15.3), the status line reads:

```
Page: ppp  Line: rrr  Col: cccc   Mode: xxxxxxxx
```

The page number 'ppp' is relative to the start of file, with the first page of a file being page 1. The line number 'rrr' is relative to the start of a page. The first data line on a page is line one. The other indications are as before.

If the 'justify right' mode is selected, the status line will have the caption "Just: R" appended.

4.4 Immediate commands

Immediate commands are key combinations that have an effect other than text entry. The effect of the command is immediately visible on the screen - with one exception ('Temporary margin').

These commands are detailed in section 6.

The general scope of immediate commands covers:

Cursor movement
Temporary margin
Deletion of text characters
Deletion of a line
Insertion of a line

Invoking help
Refreshing the screen
Repositioning/resizing the screen
Swapping the data entry mode
Garbage collection

Invoking extended commands

4.5 Extended commands

The repertoire of extended commands is intended to provide a range of facilities for repetitive processing through a file, as well as for more simple "single-shot" processes.

The method of entry of the commands is described in section 7. Detailed reference material for all extended commands is contained in section 8. The general scope of the extended commands covers:

Cursor movement
Deletion of text characters
Deletion of a line
Insertion of a line
Find and exchange string
Block definition and manipulation
Position Marker
Tabs definition and manipulation
Margins setting
Justification of text
Paragraph reforming
Case adjustment
Numbering and Sorting
Memory management
Undo
Command file processing
Miscellaneous facilities
File reading and writing
Program termination

4.6 Absent commands

There are several omissions from the command repertoire that may at first seem surprising. For example, there are no commands to show a directory of a device, to delete a file, rename a file etc.

The reason for these omissions is that the Editor runs as an independent task within the QL and is arranged such that the user may "detach" from the task, return to the 'normal' QL command environment, issue the necessary commands in the conventional way, and then return to the Editor task with the "attach" sequence. This presupposes that the Editor was initiated with the QL EXEC command and NOT the QL EXEC_W command. This sequence of events may (and usually will) result in the Editor screen being a bit of a mess when control returns to the Editor task. This mess is rectified with the "immediate" command to refresh screen.

The usual convention on the QL to detach from one task and attach to another is the simultaneous depression of the keys CTRL and C.

With a small extension of this idea, and assuming sufficient memory is available on the QL, having more than one copy of the Editor in memory at the same time can be very convenient - for example scanning a compiler listings file for error/warning lines using one copy, and having the source file available for immediate amendment in another copy. It is more for this reason than personal taste that the size/position and paper/ink attributes of the screen are under user control, so that concurrent instances of the Editor may be distinguished.

The user should take care under these circumstances that the screen mode when returning (attaching) to Editor is as it was when the user detached from Editor. Most particularly, the screen mode must not be changed from high resolution to low resolution.

4.7 Running the demos

The command file "demo_cmd" is provided to demonstrate a few of the features of Editor, and to illustrate the principles of command files.

To run the file, start Editor in the manner described above, set up a reasonable screen size - say 80 characters wide - using SHIFT/F4. Then press the F3 key to open up the command line and type:

rc/flp1_demo_cmd/ and press ENTER.

If you are greeted by a "File not found" error message, then adjust the "flp1_" in the command to whatever is appropriate for you. Otherwise, sit back and watch. Once you have had a chance to look through the extended commands described in section 8, you might care to come back to the demo_cmd file and look at how it works.

The "column_cmd" is provided for similar reasons. One difference with this file is that it provides an example of 'parameterised' command file operation. The start-up command is similar:

rc/column_cmd/ and press ENTER.

The command file will cause two messages to appear in the command window. The first message asks "Enter the number of lines per column: ". You should respond with a number, say between 10 and 20. The second message asks "Enter the width of the column: ". Again you should enter a number, this time between 25 and 30. The command file will create a series of lines in 'single column' format, and then cut and join lines so that they appear in '3 up column' format - similar to newspaper or magazine layout.

The third file "quill_cmd" does most of the donkey work in converting a Quill "_doc" file into straight text file format. As with "column_cmd", several initial questions have to be answered, after which the command file just gets on with the job.

4.8 Usage of files

Because Editor is set up as a multi-tasking program, it is important to know what files the Editor has open at any time, to allow you to change disks, tapes etc.

The answer is that Editor almost always has no files open. The particular exceptions are as follows:

- when reading in a file (during the r, ru and af commands)
- when writing a file (during the w, wr and bw commands)
- when providing on-screen help (during F1, F1/SHIFT sequence)
- when processing a command file (during the rc command)

If you give Editor the command to write a file called "fred", and then detach from Editor (using CTRL/C) into SuperBASIC and say something like:

```
        copy mdv2_fred, ser1hz
```

as an attempt to send the file to the printer (or whatever is attached to serial port 1), then the operating system may well respond with an "in use" message, indicating that Editor has not yet finished writing the file. You should wait until the Editor indicates that the command has been completed, and then retry the SuperBASIC command.

5. Terminology of text

This section is a bit detailed. It could be regarded as one big 'text box' - to be skipped on first reading of this guide.

The section is placed here since it contains the Editor's version of the definitions of many of the terms and ideas that crop up later. All of these terms have 'every day' meanings that will suffice until you need to know in more detail what is being said.

5.1 Characters

On most modern computers, and certainly the QL, a 'character' is a unit of storage that may take any value 0 through 255.

In English, we have 26 letters - 52 if you count upper and lower case as being different, and perhaps 20 symbols. It would seem that there are more 'codes' available than are needed to describe the language.

The assignment of significance of each code has been settled, and again most computers use or support the ASCII definitions. In this definition codes 0 thru 31 are control codes - typically non-display - codes 32 thru 127 are displayable, mostly alphabetic and numeric plus some symbols for punctuation, maths and so on. The codes above 127 are generally undefined by ASCII, and are also classed as non-display. On the QL, these codes result in various strange characters on screen - arrows, foreign language symbols and the like.

The only point in mentioning all this is that, in certain modes, Editor will make a distinction between "display" characters and "non-display" characters. Specifically, if a file has explicitly or implicitly been declared as a "document" (by use of the RD or MD commands), then word wrapping and paragraph reforming operate slightly differently. The non-display characters are treated by the program as though they are highlighting (print formatting) characters - for example to invoke bold or underline etc - and as such will occupy no space on the paper at print time.

ASCII character 31 (CTRL/Copyright on the keyboard) is an exception. This character is treated by Editor as a "non-break space". When Editor is required to break or restructure lines (during word wrap or paragraph reforming), a line will be broken at the occurrence of a space character, and if 'right justify' is in effect, spaces will be invented between words to pad out the line. While both of these effects are implicitly required by the user, it is sometimes the case that specific words should NOT be separated by either a line break or a certain number of spaces. If the non-break space character is used instead of the normal space character, Editor will identify each component word as separate, but will not attempt to insert spaces between the words when padding lines, nor will it put a line break within the sequence of words. The non-break space is displayed on screen as a triangle.

5.2 Words

Editor has three separate uses for 'word' related operations - cursor movement by word, deletion by word and searching by word. The first two are quite similar, and share the same definition for a word. The third uses a different definition.

The intention in Editor has been to make word related operations 'natural' - with the effect that in almost every case, ~~the~~ Editor knows what you mean by a word. Just using the commands will confirm that this is so - there is no real requirement to understand how they work.

However, if you feel you need to adjust the word delimiter strings in the configurator, you should really understand how they are used. The text box below has the details.

When moving by word, the current cursor position is significant. If the cursor is 'within' a word, the next non space-type character after a space-type character is sought (if move right word); or the next left space-type character prior to a non space-type character is sought (if move left word). If the cursor is at the start of a word, then the action is the same. If the cursor is not within a word, then the next non space-type character is sought (move right word) or the next left space-type character prior to a non space-type character (move left word).

When deleting, the current cursor position is significant. If the cursor is 'within' a word, then the residue of the word only is deleted (depending on the direction of the deletion). If the cursor is at the start of a word, then the word and trailing space-type characters are deleted (if delete right word) or the prior word and trailing spaces are deleted (if delete left word). If the cursor is positioned on a space-type character, it and trailing space-type characters are deleted (if delete right word), otherwise if delete left word, the action is similar, determined by the type of the characters preceding the space-type character.

When searching, the definition of a word is clearly a string of characters delimited by space-type characters.

In the above, reference to 'space-type' characters is meant to imply any character in the delimiter string in use (either the word movement delimiter string or the word search delimiter string). Conversely, the non space-type characters.

There is actually a fourth instance of word identification in Editor. It is used for word wrap (and paragraph reformatting) in the specific identification of non space characters, and is therefore the conventional idea of a word - being literally space delimited.

5.3 Lines

A line is usually a series of characters terminated by a 'carriage return' or 'linefeed' character, or both. For some reason, the QL seems to favour merely the 'linefeed' character, while many other systems actually use the character pair. On the QL, a 'linefeed' character is generated by pressing the ENTER key on the keyboard.

In Editor, this is not really the end of the story. Editor has the capability to process files that are not "text" or "document" files. An "unformatted" file could be very large and yet not contain a single linefeed character.

So, in Editor, a line is defined as being a series of characters which is terminated by the earlier occurrence of (a) a linefeed character or (b) the user defined maximum line length.

5.4 Paragraphs

In Editor, a paragraph is a series of non-blank lines terminated by a blank line. In the special case of a document file, a 'page break' is also seen as a paragraph terminator.

In this regard, Editor is at a disadvantage to a word-processor, since it does not have the mandate to put 'paragraph markers' into the file data.

Paragraphs are significant within Editor as follows:

when establishing the appropriate 'left margin'
when justifying lines
when reforming paragraphs
when moving the cursor to next/prior paragraph

5.5 Pages

The idea of a page is relevant only when Editor is processing a file as a 'document'. A 'page break' occurs whenever the "page throw" character (ASCII 12, obtained by pressing CTRL and L and displayed as L with a bar above) occurs IN COLUMN 1 of a line, or if the number of lines on the current page exceeds the user specified page length. In either case, Editor will display a hatched line prior to the first line of the new page. A page break caused by the occurrence of CTRL/L in column 1 is termed a 'hard' page break. A page break computed by the program - because of an excess of lines - is a 'soft' page break, since if the number of lines on the page were to be increased or reduced, the page break would move or disappear.

3.6 Blocks

Blocks are a mechanism or convenience for use while editing. The user defines the start point and the end point for a block as necessary. The block may then be copied, moved or deleted as required, and - barring deletion - as often as required.

Editor supports three different types of block - Character blocks, Column blocks and Line blocks.

A Character block is a sequence of characters from the defined start of block through to and including the defined end of block. This string of characters may be wholly contained on one line, or may span several lines. Characters on the first line of the block prior to the block start column are not part of the block. Characters on the last line of the block after the block end column are not part of the block. All characters between, including linefeed characters, are part of the block.

A Column block defines a 'rectangle' over the text. The top left corner of the rectangle is the block start column/line. The bottom right corner of the rectangle is the block end column/line. Linefeed characters are never considered to be part of a column block.

A Line block is a sequence of complete lines from the defined start of block through to and including the defined end of block. The cursor column is not taken as significant when a line block start or end is defined.

3.7 Files

Operations within Editor may be affected, depending on the type of file that Editor is processing. Almost all of the commands in Editor are applicable to all types of file. Specific exceptions are the commands related document paging - these commands are restricted to 'document' files.

The user can declare a file to be a 'text' file or an 'unformatted' file - by using the appropriate Read file command. Additionally, Editor may be instructed that a 'text' file is in fact a 'document' (as distinct from a program source file, list of names and addresses etc all of which are 'text') by virtue of certain operations the user performs on the file. Editor will never treat an 'unformatted' file as though it were a 'document'.

A 'text' file may be read using the RU (read unformatted) command. There are certain side-effects, detailed later in section 8.15, that may make this desirable. Editor will detect if an unformatted file is in fact in "QL executable file" format, and take appropriate action (when later writing the file) to preserve this format.

6. Immediate commands

Immediate commands are a family of key combinations - each being only a "single" depression - that have an immediate effect on the main screen display. Additionally, some actually have an effect on the data in the file being edited.

In the following paragraphs, the key combinations and their effect are described. Where the combination involves multiple keys, it should be understood that all keys in the combination should be pressed at the same time.

The commands have been arbitrarily grouped into the following sections:

Cursor movement, scrolling and paging
Data deletion
Line insertion
Temporary left margin
Help
Screen refresh/resize
Mode change
Garbage collection
Invoking extended commands

6.1 Cursor movement, scrolling, paging

All cursor movement commands involve the use of one (and only one) of the five keys:

UP arrow, DOWN arrow, LEFT arrow, RIGHT arrow and ENTER

On their own, the arrow keys merely move the cursor.

When used with the SHIFT key, the LEFT and RIGHT arrows cause movement by a 'word', rather than by a character.

When used with the ALT key, the LEFT and RIGHT arrows cause movement to either end of the current line.

When used with the CTRL key, the LEFT and RIGHT arrows cause deletion, so that CTRL/LEFT deletes the character to the left of the cursor, CTRL/SHIFT/LEFT deletes the word to the left of the cursor, and CTRL/ALT/LEFT deletes the whole of the line prior to the cursor etc.

In the following command descriptions, frequent use is made of the words "logical" and "physical". This hedging arises because, when processing a document file, screen lines may display page breaks and the cursor may never 'land' on a page break. Also 'non display' characters on a line may cause the physical column to differ from the logical column.

6.1.1 Cursor up

Key(s): UP arrow
Effect: The cursor moves to the logically prior line in the file, holding the same physical column position. If necessary the "window" scrolls down.

Data: Not affected .
Errors: Top of file (cursor is already at top).

6.1.2 Cursor down

Key(s): DOWN arrow
Effect: The cursor moves to the logically next line in the file, holding the same physical column position. If necessary the "window" scrolls up.

Data: Not affected
Errors: End of file (cursor is already at end).

6.1.3 Cursor left

Key(s): LEFT arrow
Effect: The cursor moves one character to the left on the current line in the file. If necessary the "window" pans right. No effect if the cursor is at column 1 of the line.

Data: Not affected
Errors: None

6.1.4 Cursor right

Key(s): RIGHT arrow
Effect: The cursor moves one character to the right on the current line in the file. If necessary the "window" pans left. No effect if the cursor is at the maximum line length.

Data: Not affected
Errors: None

6.1.5 Word left

Key(s): SHIFT/LEFT arrow
Effect: The cursor moves to the column to the right of the prior word on the current line. If there is no prior word on the line, the cursor moves to column one of the line. If the cursor is already at column one of the line, the cursor moves to the logical end of the prior line. If necessary the "window" pans right, or left, or scrolls down. See section 5.2 - "words"

Data: Not affected
Errors: Top of file, if at column 1 of line 1

6.1.6 Word right

Key(s): SHIFT/RIGHT arrow
Effect: The cursor moves to the first character of the next word to the right on the current line. If there is no subsequent word on the line, then the cursor moves to column one of the next line. If necessary the "window" pans left, or right, or scrolls up. See section 5.2 - "words".

Data: Not affected
Errors: End of file, if at end of last line of file.

6.1.7 Cursor to start of next line

Key(s): ENTER (in Overstrike mode)
Effect: The cursor moves to column one of the next line in the file. If necessary the "window" pans right, or scrolls up. If no further lines exist on the file, then a new line is created. A 'soft' page break may be caused.

Data: Not affected, unless at end of file
Errors: Out of memory

6.1.8 Cursor to start of current line

Key(s): ALT/LEFT arrow
Effect: The cursor moves to column one of the current line in the file. If necessary the "window" pans right.

Data: Not affected
Errors: None

6.1.9 Cursor to end of current line

Key(s): ALT/RIGHT arrow
Effect: The cursor moves to the "logical" end of the current line in the file. The logical end of the line is the character position to the right of the rightmost actual character on the line. If necessary the "window" pans left.

Data: Not affected
Errors: None

6.1.10 Cursor to top of screen

Key(s): SHIFT/UP arrow
Effect: The cursor moves to column one of the first data line on the screen. If necessary the "window" pans left.

Data: Not affected
Errors: None

6.1.11 Cursor to bottom of screen

Key(s):	SHIFT/DOWN arrow
Effect:	The cursor moves to the logical end of the last data line on the screen. The logical end of line is the character position to the right of the rightmost actual character on the line. If necessary the "window" pans left or right.
Data:	Not affected
Errors:	None

6.1.12 Scroll up

Key(s):	ALT/UP arrow
Effect:	The cursor remains on the same logical line of the file, and in the same column position, but the file data on display is shifted toward the top of file. Consequently, the line containing the cursor appears to move down the screen by one line. If the cursor is on the bottom line of the screen, then the command causes the cursor to move to the same physical column position on the logically prior line of the file. The command is ignored if the top of file is the first line on screen.
Data:	Not affected
Errors:	None

6.1.13 Scroll down

Key(s):	ALT/DOWN arrow
Effect:	The cursor remains on the same logical line of the file, and in the same column position, but the file data on display is shifted toward the end of file. Consequently, the line containing the cursor appears to move up the screen by one line. If the cursor is on the top line of the screen, then the command causes the cursor to move to the same physical column position on the next data line of the file. The command is ignored if the cursor is on the last line of the file.
Data:	Not affected
Errors:	None

6.1.14 Page up

Key(s): SHIFT/ALT/UP arrow

Effect: The cursor remains in the same relative position on the screen, and the file data on display is shifted toward the top of file. The screen is (in principle) repainted. The physical cursor column will never move, but the cursor may change line if paging violates top of file - cursor goes to first line - or if by remaining on the same line would 'land' on a page break - cursor moves to the line before the page break.

Data: Not affected

Errors: Top of file

6.1.15 Page down

Key(s): SHIFT/ALT/DOWN arrow

Effect: The cursor remains in the same relative position on the screen, and the file data on display is shifted toward the end of file. The screen is (in principle) repainted. The physical cursor column will never move, but the cursor may change its physical line on the screen if paging violates end of file - cursor goes to last line - or if by staying on the same line would 'land' on a page break - cursor moves to the line after the page break.

Data: Not affected

Errors: End of file

6.1.16 Paragraph up

Key(s): CTRL/ALT/UP

Effect: The cursor moves to column 1 of the line commencing the current paragraph - or the prior paragraph, if the cursor is already on the first line of the current paragraph.

Using Editor's definition for a paragraph, this means that after the command, the cursor will be either on the top line of the file, or a blank line or page break will immediately precede the line containing the cursor.

If the target line is visible on the screen before the command, the cursor will move to that screen line. Otherwise, the screen will be repainted such that the cursor remains on the same physical screen line.

Data: Not affected

Errors: None

6.1.17 Paragraph down

Key(s): CTRL/ALT/DOWN

Effect: The cursor moves to column 1 of the line commencing the next paragraph.

Using Editor's definition for a paragraph, this means that after the command, the cursor will be either on the bottom line of the file, or a blank line or page break will immediately precede the line containing the cursor.

If the target line is visible on the screen before the command, the cursor will move to that screen line. Otherwise, the screen will be repainted such that the cursor remains on the same physical screen line.

Data: Not affected

Errors: End of file

6.2 Data Deletion

6.2.1 Delete character left

Key(s): CTRL/LEFT arrow
Effect: The character to the left of the cursor is removed from the current line, and the cursor consequently moves one column to the left on the current line in the file.

If the cursor is on column 1 of the line, then the current line is appended to the prior line of the file. If necessary the "window" pans right, left, or scrolls down.

If the cursor is beyond the logical end of the current line, the cursor is moved to the character position to the right of the logical end of line, but no deletion occurs.

Data: Affected
Errors: Top of file, if at line 1, column 1
Line too long, if this and prior line exceed max length

6.2.2 Delete character right

Key(s): CTRL/RIGHT arrow
Effect: The character under the cursor is removed from the current line. The cursor remains in the same relative position. The command is ignored if the cursor is at or beyond the logical end of the line.

Data: Affected
Errors: End of file, if at/beyond end of last line of file

6.2.3 Delete word left

Key(s): CTRL/SHIFT/LEFT arrow
Effect: The word to the left of the cursor is removed from the current line, and the cursor consequently moves a number of columns to the left on the current line in the file. If the cursor is on column 1 of the line, the current line is appended to the prior line of the file. If necessary the "window" pans right, left, or scrolls down.

If the cursor is positioned beyond the logical end of line, the cursor is moved to the character position to the right of the end of line, but no deletion occurs.

The precise action of this command is determined by the placement of the cursor - either at the start of a word, or somewhere within a word, or not within a word. Each starting condition produces a slightly different effect. For more detail, see section 5.2 - "words"

Data: Affected
Errors: Top of file, if at line 1, column 1
Line too long, if this and prior line exceed max length

6.2.4 Delete word right

Key(s):	CTRL/SHIFT/RIGHT arrow
Effect:	The word under the cursor is removed from the current line. The cursor remains in the same relative position. If the cursor is at or beyond the logical end of the line, the following line of the file is appended to the current line, at the cursor position.
	The precise action of this command is determined by the placement of the cursor - either at the start of a word, or somewhere within a word, or not within a word. Each starting condition produces a slightly different effect. For more detail, see section 5.2 - "words"
Data:	Affected
Errors:	End of file, if at/beyond end of last line of file
	Line too long, if this and next line exceed max length

6.2.5 Delete to start of line

Key(s):	CTRL/ALT/LEFT arrow
Effect:	The characters to the left of the cursor are removed from the current line, and the cursor consequently moves to column one of the current line in the file. The command is ignored if the cursor is on column 1 of the line. If the cursor is beyond the logical end of line, the command causes the cursor to move to the logical end of line, but no deletion occurs. If necessary the "window" pans right.
Data:	Affected
Errors:	None

6.2.6 Delete to end of line

Key(s):	CTRL/ALT/RIGHT arrow
Effect:	The characters under and to the right of the cursor are removed from the current line. The cursor remains in the same relative position. The command is ignored if the cursor is at or beyond the logical end of the line.
Data:	Affected
Errors:	None

6.2.7 Delete line

Key(s):	CTRL/SHIFT/ALT/LEFT arrow
Effect:	The line containing the cursor is removed from the file. The cursor remains in the same relative position, but the file data is shifted up toward the top of file, with the effect that the cursor is on the next relative line of the file. If the cursor is on the last line of the file, the cursor moves back a line. A deleted line may be recalled using the Undo Delete command.
Data:	Affected
Errors:	None

6.3 Line insertion

6.3.1 Insert new line

Key(s): CTRL/DOWN arrow
Effect: A new line is inserted logically immediately below the current position of the cursor, irrespective of the current cursor column. The cursor moves to the 'first' column of the new line - the first column is as defined by the left margin, temporary left margin or indent margin - see 8.6.1/2. The whole of the new line is blank. If necessary the "window" pans right, or scrolls up.

Data: Affected
Errors: Out of memory

6.3.2 Split current line

Key(s): ENTER (Insert mode)
Effect: A new line is inserted logically immediately below the current position of the cursor. The character under the cursor and any characters to the right of the cursor are removed from the current line. The cursor moves to the 'first' column of the new line - which is as defined by the left, temporary left or indent margin - see 8.6.1/2. The new line contains the characters removed from the original line. If necessary the "window" pans right, or scrolls up.

Data: Affected
Errors: Out of memory

6.3.3 Word wrap

Key(s): Automatic
Effect: If the user enters a 'data' character onto the current line in a column position logically beyond the right margin (see 8.6.3), the line is automatically split in such a way that the right margin violation is removed. In this context, 'data' characters do not include SPACE and TAB. The split takes place at the start of a 'word' (see 6.3.2 for the split method), but the cursor remains in the same position relative to the text being entered. The first non-space character on the new line will occur in the column defined as the left or temporary left margin - see 8.6.2. If necessary, the "window" pans right, and/or scrolls up.

Data: Affected
Errors: Out of memory

6.3.4 Insert line at end of file - see 6.1.7

6.4 Temporary Left Margin

Key(s): ALT/TAB

Effect: The cursor position at the time of the key depression is taken as indicating the column to be used as the left hand margin. This temporarily overrides the settings for 'indent' margin and 'left' margin.

The temporary setting remains in force until the cursor is moved out of the paragraph, or is moved to a point in the current paragraph logically prior to the line and column at which the temporary margin was set.

This facility enables the easy use of "hanging" paragraphs - such as the current text. The word "Effect:" above is the default left margin, while the left margin presently in force has been set with ALT/TAB.

Normal cursor movement is allowed within the 'indented' paragraph, with the rider that if the cursor is moved to a position logically prior to the point at which the ALT/TAB was pressed, then the 'real' left margin again comes into effect. Equally, if the cursor is moved down the document out of the current paragraph, the temporary margin is released.

So, the feature is implemented only as an aid to text insertion, pretty much like the TAB key itself. No routine (in Editor) remembers that a TAB was pressed - a TAB is immediately converted to a series of one or more spaces.

Data: Not affected (directly)

Errors: None

6.5 Help

Key(s): F1 or F1/SHIFT

Effect: The screen is completely cleared and the on-line help file is accessed. If the file is not found on the default device, a message is displayed and the main screen is restored. If F1 was used, the first page of the help text is displayed - this contains the "immediate" command help. To return to the main screen, press ESC. To move to the next page, press any other key. The remaining help text concerning the "extended" commands is displayed in the same way. The main screen is then restored.

Data: Not affected

Errors: Help information not available

6.6 Data entry mode

Key(s):	F5
Effect:	The data entry mode is changed. The data entry mode is one of "overstrike" and "insert" and the current mode is displayed on the status line. When in overstrike mode, a character entered from the keyboard replaces the character under the cursor. When in insert mode, a character entered from the keyboard is inserted in the line prior to the character under the cursor. The current mode is permanently displayed on the status line.
Data:	Not affected
Errors:	None

6.7 Screen refresh

Key(s):	F4
Effect:	The screen is repainted, refreshing the border, main screen and status line.
Data:	Not affected
Errors:	None

6.8 Screen size/position

Key(s):	F4/SHIFT
Effect:	The screen is cleared and the border repainted. Using the arrow keys with or without the ALT key, the user may resize and/or reposition the screen. The process stops when the ENTER key is pressed. The minimum width of the screen is 20 characters. The minimum height of the screen is 4 lines. During the process, the coordinates of the top leftmost point of the screen, the screen width and the screen depth are displayed.
Data:	Not affected
Errors:	None

6.9 Invoke extended commands

6.9.1 Invoke new command

Key(s): F3
Effect: The status line is cleared and the command line editor is invoked. No commands are shown in the command line. See section 7 for further action.

Data: Not affected by immediate command
Errors: None

6.9.2 Edit and execute last command group

Key(s): F3/SHIFT
Effect: The status line is cleared and the command line editor is invoked. The current commands (those last specified using F3 or F3/SHIFT) are made available for editing in the command line. See section 7 for further action.

Data: Not affected by immediate command
Errors: None

6.9.3 Re-execute last command group

Key(s): F2
Effect: The status line is cleared and the current command group (as specified using F3 or F3/SHIFT) is shown in the command line. The commands are processed. See section 7 for further action.

Data: Not affected by immediate command
Errors: None

6.9.4 Edit and execute last find/exchange command

Key(s): F2/SHIFT
Effect: The status line is cleared and the command line editor is invoked. The current "find/exchange" command is made available for editing in the command line. See section 7 for further action.

Data: Not affected by immediate command
Errors: None

6.9.5 Re-execute last find/exchange command

Key(s): CTRL/F2
Effect: The status line is cleared and the current "find/exchange" command is shown in the command line. The command is processed. See section 7 for further action.

Data: Not affected by immediate command
Errors: None

6.10 Terminate extended command

Key(s):	ESC
Effect:	The command in progress (if any) is immediately aborted. If the command was a "multi-line" command - e.g. "renumber" or "block insert", the cursor is moved to the last line processed by the command, otherwise the cursor does not move. If no command is in progress, ESC is treated as data.
Data:	If command in progress, not affected by immediate command
Errors:	Commands abandoned

6.11 Garbage collection

Key(s):	F5/SHIFT
Effect:	The internal data and control tables are tidied up. Where a significant amount of editing, block shifting and the like have taken place, this will have the effect of slightly improving program response. Under various circumstances, usually involving shortage of memory, the Editor will collect garbage automatically.
Data:	Not affected
Errors:	Temporary message - "Collecting garbage"

7. Extended commands

The extended commands are those commands accessed by pressing the F3 function key. When the F3 key is pressed, a 'command window' opens at the bottom of the Editor screen - replacing the status line - and the command line editor is ready to accept input.

The command line editor will accept a single command, or several commands or repeated groups of commands in various combinations.

Each command consists of an alphabetic mnemonic, of one or two characters. Editor supports many such commands - about 100 - in several families.

The general syntax of the command line is described in the remainder of this section, and the following section deals with each of the extended commands in detail.

Having established a command (initiated by the F3 key), other function key combinations may be used to repeat the last command, recall the last command etc - as described in section 6. Several of these function key commands also access the command line editor.

7.1 Command line editor

The command 'line' is in fact a window that can grow upwards as necessary, depending on the complexity (length) of the commands entered. The window will stop growing when only one line of the main screen remains visible, or if the length of the command equals the specified maximum length of a line, whichever is sooner.

When entering commands, the Help function remains available (keys F1 and F1/SHIFT).

The command line editor supports most, but not all, of the edit functions of the main screen. Particular omissions are the 'word' related functions - move word, delete word. Additionally, the UP arrow and DOWN arrow position the cursor at start and end of line respectively.

To exit from the command line editor WITHOUT executing any commands, the ESC key is used. To pass the command line for processing, the ENTER key is used.

7.2 Command format

All commands are identified by a one or two character alphabetic mnemonic.

Some commands require additional information to qualify the required action. For example, the "read file" command 'R' must be followed by a string - the file name of the required file. The "exchange" command 'E' is, usually, followed by two strings - the first is the string to be matched and the second is the replacement string. Some other commands must be followed by a number. For example, the "set tab interval" command 'ST' must be followed by a number - the tab interval. Yet others require no qualification - the "go to Top" command 'T' (or 'GT') is self sufficient.

Any command may be prefixed by a number. This is interpreted as being a repeat count. For example, the "cursor right" command 'CR', which requires no qualification, could be written as '10cr' to move the cursor 10 characters to the right.

More than one command may be given on one command line. Successive commands are separated by either a space or a semicolon. Commands are always processed left to right. So, to return to the top of file and then exchange the first occurrence of "fred" to "joe", the command could be either

```
T E"fred"joe"      or      T;E"fred"joe"
```

Generally, any spaces between a command and a trailing string or number will be ignored, so T; E "fred"joe" would be ok.

Several commands may require to be grouped together - in order to assign a repeat count to the group as a whole. This is done by putting brackets around the commands to be grouped.

A quick way of generating lots of data in an initially empty file is as follows:

```
8 (ce; e//...abcde/); 3 (t;bs;b;be;3(b;bi))
```

This will repeat the operation "cursor to end of line, exchange a null string with '...abcde'" eight times - resulting in a line 64 characters long. That done, the top line of the file is marked as the start of block, the bottom line of the file is marked as end of block, and the block so defined is inserted three times at end of file. This sequence is repeated three times. The first time, there is only one line in the block - the second time there are four lines etc. When the command is complete, the file contains 64 lines of 64 characters.

A similar effect could be achieved by

```
8e//...abcde/;bs;be;63bi
```

[illegible]1 Strings in the command line

As may be inferred from the above examples, where strings are to be specified in a command line, they must be "delimited". The delimiter at the start of the string must be repeated at the end of the string. All characters between the delimiters are considered part of the string.

For reasons of syntax, certain characters may NOT be used as delimiters. These are:

All letters
All numbers
The characters space, semicolon, left and right bracket.

As will be seen later, Exchange has a 'simple' and a 'compound' form. In the 'simple' case of "exchange", where two strings are usually required, there is a convention that the 'middle' delimiter serves as the end delimiter of the first string and the start delimiter of the second string.

If a string is the last item on the command line, then it is permissible to omit the closing delimiter.

7.2.2 Special characters

Some characters with ASCII codes above 127 may not be entered into the file directly - because they are interpreted as immediate commands. Such characters can however be entered via the command line.

The QL User Guide indicates the key combinations required to achieve each required code.

Any character which the command line editor is specifically looking for - e.g. ESC, F1, F1/SHIFT - may be entered by using the ALT key in combination - e.g. ESC/ALT.

All Special characters may be entered via the command line, including characters and combinations that are normally used for editing the command line. This is done by using the sequence ALT/ESC followed by the character required. The characters entered on the command line will be ALT key (value 255) and the ASCII value of the combination used - eg; left arrow (192). The 255 character is entered in case it is the character required. If it is not, then delete it.

7.3 Command termination

Once a command line has been put into execution, it will stop for one of three reasons:

The commands have been accomplished
The program has detected an error condition
The user aborts the commands (ESC key)

In the latter two cases, an error message will be displayed, displacing the status line, giving the reason for command termination. Typical examples are: "Search failed", "End of file", etc. If the ESC key is pressed, the error message is "Commands abandoned".

7.4 Status line during command execution

If a command line includes commands that result in cursor movement, then the display of the command line is overwritten by the status line during execution of the commands. To indicate that a command is still in progress, the first character of the status line is the copyright symbol.

If the commands involve no explicit cursor movement, the command window will usually remain on display until command termination. Exceptions are when reading or writing files, when a confirmation message is displayed.

If a complex, or potentially lengthy, command (sequence) has been given, typically involving use of the RP (repeat) command, command processing can either be left to continue, or depending on circumstances, it can be put into one of two "go faster" modes.

Each time a command results in the movement of the 'cursor' - in this case the current position in the file, since the cursor will not be displayed during command execution - the status line is updated to reflect the new position. If this is not needed, then command processing will speed up somewhat if it is disabled. You do this simply by pressing a 'harmless' key - for example LEFT ARROW.

If the effect of the command also includes updating of the main screen, this too can be disabled. Press the chosen key a few more times - say 8 or 9 - and you will notice that the main screen area ceases to change. You are now "flying blind", but can take it as read that the command processing is proceeding at top speed.

The speeding up effect is almost always very considerable.

Even though several characters may have been typed - and clearly they have been 'seen' by Editor - the ESC key is still available if it should be required to terminate the command processing.

8. Extended commands - Reference

This section lists all of the extended commands supported by Editor. It is intended that this should be a reference section. That is, to say that it is not meant to be read from top to bottom in the manner of a novel or article - you may find it heavy going to plough through the whole section in this manner. The purpose is to note precisely the syntax and effect of each of the commands.

The commands are dealt with in 'functional groups', as follows:

Cursor movement commands
Block commands
Find and exchange
Insert and Delete
Position Marker
Tabs and Margins
Justification
Paragraph reforming
Case adjustment
Numbering and Sorting
Memory management
Undo
Command files
Miscellaneous commands
File handling
Terminate program commands

In each case, the same headings are used to describe the commands.

```
Command:                Example:
 Qualif:
 Effect:
   Data:
 Errors:
```

The command mnemonic is given, followed by an example or two. The line below shows the qualifiers (if any) that may be used with the command. The main description of the command is given under the heading 'Effect', and this section will vary from one line to several paragraphs depending on the complexity and power of the command. The heading 'Data' details whether the file data is affected, and under what circumstances, and finally a note is given of what error message may conceivably occur during the processing of the command.

8.1 Cursor movement commands

8.1.1 Cursor to next line

```
Command: N                Example: N
 Qualif: None
 Effect: Moves cursor to column 1 of the logically next line

   Data: Not affected
 Errors: End of file
```

8.1.2 Cursor to prior line

```
Command: P                Example: P
 Qualif: None
 Effect: Moves cursor to column 1 of the logically prior line

   Data: Not affected
 Errors: Top of file
```

8.1.3 Cursor to named line

```
Command: GL               Example: GL 482
 Qualif: Number mandatory
 Effect: Moves cursor to column 1 of the indicated line.  If the
         named line is not a data line - is a page break - the
         cursor is moved to the line before or after the named
         line, depending on whether the movement is forward or
         backward relative to the current line.

   Data: Not affected
 Errors: End of file; Number expected
```

8.1.4 Cursor to top of file

```
Command: GT               Example: GT        Synonym: T
 Qualif: None
 Effect: Moves cursor to column 1 of the first data line

   Data: Not affected
 Errors: None
```

8.1.5 Cursor to end of file

```
Command: GB               Example: GB        Synonym: B
 Qualif: None
 Effect: Moves cursor to column 1 of last line of file. Where
         possible, the last line of the file is displayed as the
         middle line of the screen.

   Data: Not affected
 Errors: None
```

8.1.6 Cursor to Start of line

```
Command: CS              Example: CS
 Qualif: None
 Effect: Moves cursor to column 1 of the current line

   Data: Not affected
 Errors: None
```

8.1.7 Cursor to End of line

```
Command: CE              Example: CE
 Qualif: None
 Effect: Moves cursor to 1 column beyond the current line length

   Data: Not affected
 Errors: None
```

8.1.8 Cursor right

```
Command: CR              Example: CR
 Qualif: None
 Effect: Moves cursor one column to the right on the current line.
         Command ignored if cursor position exceeds the maximum
         line length. If necessary, the "window" pans left.

   Data: Not affected
 Errors: None
```

8.1.9 Cursor left

```
Command: CL              Example: CL
 Qualif: None
 Effect: Moves cursor one column to the left on the current line.
         Command ignored if cursor at column 1. If necessary, the
         "window" pans right.

   Data: Not affected
 Errors: None
```

8.1.10 Cursor right word

```
Command: CW              Example: CW
 Qualif: None
 Effect: Moves cursor one 'word' to the right.  If the cursor is
         beyond the logical end of the current line, the cursor
         moves to column 1 of the next line.  If necessary, the
         "window" pans left, or right and/or scrolls up.  See
         section 5.2 - "words".

   Data: Not affected
 Errors: End of file
```

8.1.17 Cursor to start of Paragraph

```
Command: CP               Example: CP
 Qualif: None
 Effect: The cursor is moved to column 1 of the line which
         commences the next paragraph (i.e. a non blank line
         following a blank line or page break) or to the last
         column of the last line of the file.

         This extended command corresponds to the immediate
         command 'cursor to start of paragraph' - obtained using
         CTRL/ALT/DOWN arrow.  The prior paragraph may be obtained
         using the complementary CTRL/ALT/UP.

   Data: Not affected
 Errors: End of file
```

8.1.18 Cursor to named character

```
Command: GC               Example: GC 49352
 Qualif: Number mandatory
 Effect: Moves cursor to the indicated character position, relative
         to the start of the file. The first character of the file
         is character 1.

   Data: Not affected
 Errors: End of file; Number expected
```

8.1.19 Cursor to next or named Page

```
Command: GP               Example: (1) GP 35  (2) GP
 Qualif: Number optional
 Effect: This command is only meaningful for a 'document' file.

         Moves cursor to the start of the indicated page. If the
         page number is specified, then that page is selected.  If
         no number is specified, the next page is selected.

   Data: Not affected
 Errors: End of file
```

8.1.20 Cursor to next 'soft' Page

```
Command: GPS              Example: GPS
 Qualif: None
 Effect: This command is only meaningful for a 'document' file.

         Moves cursor to the start of the next 'soft' page. This is
         a page without an explicit page break entered by the user.
         The command is therefore useful for locating pages that
         have become 'disorganised' and possibly need repaging.

   Data: Not affected
 Errors: End of file
```

8.1.20 Cursor to prior Page

```
Command: GPB            Example: GPB
 Qualif: None
 Effect: This command is only meaningful for a 'document' file.

         Moves cursor to the start of the prior page.

   Data: Not affected
 Errors: Top of file
```

8.2 Block commands

General commentary on the different types of blocks supported by Editor was given in section 5. Some of the points made in this section will cover the same ground, but in slightly more detail. After a general introduction to 'blocks', the block commands offered by Editor will be dealt with individually.

There are three types of block. The Character Block, the Column Block, and the Line Block.

The Character Block is (almost) identical to the type of block that Quill/WordStar etc support. All characters logically contained between the start mark and the end mark inclusive are considered to be in the block, regarding each line as text to be read from left to right, progressing helically from one line to the next lower line. The implicit newline character at the end of the last line of the block is NOT considered to be part of the block. The newline character between two lines - each of which is wholly or partly contained in the block - IS considered to be part of the block.

The Column Block looks and handles differently. A column block defines a 'rectangle' of text, and so the text contained within the block is not "continuous". A column block never contains newline characters. However, moving a column block to the end of a file will cause extra lines to be created in the file. A column block could for example be defined as columns 23 to 56 of each line between line 12 and line 30. If any of the lines 12 to 30 are less than 56 characters long, they are assumed to have trailing spaces. If the cursor is moved to, say column 40 of line 200, the block could be inserted at this point (or moved to this point). What would happen is that the 19 lines 200 to 218 would be "cut" at column 39 (with spacefilling if necessary on lines shorter than 39 characters). An image of line 12 cols 23 to 56 is then pasted into line 200 at col 40 and the text of line 200 from col 41 onwards (if any) pasted behind that - therefore now occurring at column 74 onwards. The text from line 13 goes into line 201 etc etc.

The Line Block is organised such that the whole of the start line and end line of the block are considered to be within the block, irrespective of the cursor column position when the block start/end were marked. The whole of any line between the block start line and the block end line is also considered to be a part of the block. Even though the column position of the cursor is not significant when defining the Start/End positions, the column position is noted as well as the line position. This has two effects. First it means that the Cursor to 'blockmark' commands behave in the same way whether a Character block, Column block or Line block has been defined - the cursor is returned to the actual character position at which the Start/End mark was set. Even so, the Line block is still considered to start at column 1 of the (start) line and end at the last character of the (end) line and includes an implicit line feed. Second, it means that column information is available should you wish to alter the block type.

You can select the type of block that you want to use with the BT (Block Type) command. This command has a single letter qualifier - C or K or L.

BTC - Block Type Character
BTK - Block Type Kolumn
BTL - Block Type Line

Editor will also allow you to Hide or show the block, as you choose. The

BH - Block Hide

command will display the block in inverse ink/paper if it is hidden, or restore normal ink/paper if it is on display.

The 'normal' commands are fully implemented for all block types:

BI - Block Insert
BD - Block Delete
BM - Block Move

There is no Copy command. The Insert command leaves the block defined where it is - the definition does NOT move to the Insert point.

You may (rarely) need to be careful about things, otherwise you may be surprised by results. In principle, all of the rules apply equally to each block type but details of implementation differ slightly from one type to another.

For example, with block type Line, if the cursor is on the last line of the block, the BI command is valid and meaningful - insert a copy of the block immediately after the block. Even though the cursor is within the block, the user meaning is clear. In the same condition, the BM command would be pointless - effectively asking for the block to be moved to where it is.

However, if the cursor is inside a Character block, the BI/BM commands are not meaningful.

For a Column block, it depends... If the cursor is anywhere inside the block, the BI command is not meaningful. If the cursor is on the first line of the block, the BM command is not meaningful. If the cursor is anywhere else within the block, it is regarded as being on the leftmost column of the block, and the block will be moved such that the first line of the block is positioned under the cursor.

One final caveat concerning Character and Column blocks.

Editor allows you to define the block start, block end in whichever order you choose. Generally, if the (new) block start is defined 'below' - later in the file than - the (old) block end, the old block end definition is discarded. And vice versa.

Having defined, say, a Line block, you may change to a Character block (using the command BTC). This will not (normally) result in anything surprising. If the block is on display, you may see the inverse ink/paper pattern change somewhat.

Assume, however, that the Line block was defined with block start at the END (right) of a line and block end at the START (left) of the same line. That is a valid definition for a Line block - which is not column sensitive. But converting to Character block causes Editor to swap the start/end definitions, because otherwise the block is invalidly defined. Note that this ONLY APPLIES where the line block is one line - i.e. the block start and block end marks are on the same line.

Similar comments apply to Column blocks, with the addition that it is ALWAYS true that the start column must be less than or equal to the end column and the start line must be less than or equal to the end line. If a block type command is issued requesting Column blocks, the existing start and end column points will be swapped if the end column is less than the start column.

8.2.1 Mark start of block

```
Command: BS             Example: BS
 Qualif: None
 Effect: Saves the current cursor line/column position as a
         definition of the start of block point. If the current end
         of block line is earlier in the file (nearer the start of
         file), the block end becomes undefined.

         If block end remains defined the following additional
         checks apply:
         If the block type selected is Column block OR the block
         start and block end lines are equal, then if the start
         column is greater than the end column, the start and end
         columns are swapped.

   Data: Not affected
 Errors: None
```

8.2.2 Mark end of block

```
Command: BE             Example: BE
 Qualif: None
 Effect: Saves the current cursor line/column position as a
         definition of the end of block point. If the current start
         of block line is later in the file (nearer the end of
         file), the block start becomes undefined.

         If block start remains defined the following additional
         checks apply:
         If the block type selected is Column block OR the block
         start and block end lines are equal, then if the start
         column is greater than the end column, the start and end
         columns are swapped.

   Data: Not affected
 Errors: None
```

8.2.3 Insert block

Command: BI   **Example:** BI
Qualif: None
Effect: Character block:
The current line is 'broken' at the cursor point. The first (or only) line of the block is appended to the stub of the current line. If the block consists of more than one line, the amended current line is stored and all block lines saving the last are inserted. The tail of the current line (broken in the first operation) is appended to the last (or only) line of the block.

Column block:
The cursor point is taken as defining the insertion column to be used through successive lines in the file, up to the number of lines in the block. Lines that do not exist (end of file) are created. Lines which are shorter than the insertion column are padded with spaces. From the line containing the cursor, successive lines of the block are appended to the broken file lines, and the excess text of the file lines (if any) is appended.

Line block:
The lines currently defined as the block are inserted between the line containing the cursor and the following line (if any).

All types:
The definition of the block is not changed.
The cursor does not move.

Data: Affected
Errors: Block not defined; Cursor inside block; Out of memory

8.2.4 Move block

Command: BM   **Example:** BM
Qualif: None
Effect: All types:
The initial action is as described for the BI command.

The text defined as the block is deleted and the definition of the block is changed to reflect the new block position.

The position of the cursor does not (logically) move, but physically the cursor may change both line and column depending on the block type and the relative position of the block before the move.

A multi-line character block may not be moved to a point 'earlier' on the line on which it already starts.

Data: Affected
Errors: Block not defined; Cursor inside block; Block overlap
Out of memory

8.2.5 Delete block

Command: BD Example: BD
Qualif: None
Effect: The lines currently defined as the block are deleted from the file. The block becomes undefined. No cursor movement occurs unless the cursor is presently inside the block.

Data: Affected
Errors: Block not defined

8.2.6 Write block

Command: BW Example: BW.flp1_fragment_bas.
Qualif: Mandatory file name string
Effect: For the purpose of this command, all blocks are considered to be block type Line.

The lines currently defined as the block are written to the named file or device (e.g. printer). If the file already exists, confirmation is requested before the file is overwritten. The definition of the block is not changed. See also 8.15.7

Data: Not affected
Errors: Block not defined; Can't open file

8.2.7 Block Type

Command: BT Example: BTC
Qualif: Mandatory character; C, K, L
Effect: The current block type is changed, according to the chosen qualifier:

C = Character
K = Column
L = Line

In the first two cases, if the block start and block end are already defined, a check is made that the start/end columns are correctly aligned. If not, the start/end columns are swapped.

Data: Not affected
Errors: None

8.2.8 Block Hide

```
 Command: BH               Example: BH
  Qualif: None
  Effect: The 'block display' status is inverted.  If the block
          display status is set to "hide" it becomes "show", and
          vice versa.

          If a block is currently defined, and it maps onto the
          current window, the block will be repainted.

          If the block display status is "hide", the block is not
          visually disctinct on the screen - but all block
          operations are still valid and effective.

          If the block display status is "show", the block - if it
          maps onto the current window - is shown in 'inverse
          colours'.  That is, the usual ink colour is used as paper
          and the usual paper colour is used as ink.

    Data: Not affected
  Errors: None
```

8.3 Find and Exchange commands

8.3.1 General

There are two string search commands - Find and Exchange. Both commands may be qualified by search type. The search type may be one or more of:

B - search backwards
W - the search string must occur as a 'word'
C - search distinguishes upper and lower case text

In the case of Exchange, an additional option is Q - query after finding a match but before the exchange.

The command must begin with either F or E. The sequence of the qualifiers is immaterial, but there must be no spaces between the letters.

For example, FB, FWB, FWC, EBQ, ECWQ are all ok.

BF, E Q, FE are all wrong.

8.3.2 Find and Exchange - simple form

After the command has been specified, the next item is the string to be sought. This is as usual a delimited string. It may be separated from the command by one or more spaces.

For the Find command, no further specification is necessary.

For the Exchange command, the replacement string follows immediately after the closing delimiter for the search string, and is in turn ended with the same delimiter.

A minor variation is allowed if the Find/Exchange command is the last command on the command line - the terminating delimiter need not be specified; it is assumed.

8.3.3 Defaults

If the Find command is not followed by a search string, the last specified search string is assumed.

If the Exchange command is not followed by a string, the last specified search and replacement strings are assumed. If the command is followed by only one string, it is assumed to be the replacement string, and the last specified search string is assumed.

In a complex edit session, one can lose track of all these defaults. One solution is not to use them. Another is to use the "show status" command (see 8.14.5), where the current values for the strings are displayed. Yet another option (and usually the quickest, safest and probably what is required) is to use F2/SHIFT to recall the last Find/Exchange command to the command line for editing, or at least approval. For those with absolute confidence, F2/CTRL will recall AND execute the last Find/Exchange command.

These features are to be distinguished from (plain) F2, which causes the whole of the last command line to be recalled and executed, and F3/SHIFT which recalls the last command line for editing prior to execution.

8.3.4 Command execution

The command executes from the point of the cursor when the command is given. In the case of Find, the first character examined is one character to the right/left of the cursor (depending on the direction of search) - this enables successive Find commands to operate properly. In the case of Exchange, the character under the cursor is always included in the search. Irrespective of the direction of search, a replacement string which includes the search string will not be replaced again, within the same search.

If the command terminates normally, the Status line is updated to show the new cursor position. If the search string is not found, then the error message "Search failed" is displayed, and the cursor stays put.

When an Exchange takes place, a highlight block is rapidly displayed on screen to indicate where the action is happening. This is useful when multiple exchanges take place in one command (e.g. global replacement using the 'RP' command). If the 'Q' qualifier has been used, the program suspends after highlighting the initial character of the matched string. The prompt "Type Y to confirm: " is displayed on the Error line - any response other than 'y' or 'Y' is interpreted as "no".

8.3.5 Word searching

The Editor uses a fairly precise definition of a 'word' when searching strings. This definition is different (for good reason) from the definition of a 'word' when using "word left" or "delete right word" commands (see section 6). The definition (in both cases) is under user control, via the Configurator program. These definitions should be changed only with considerable care. See section 5.2 - "words".

A trivial example of the use of word search/replacement is:

```
ew.and.but.
```

In this example, an occurrence of "and" in the text file is replaced with "but" if and only if the three letters "and" are bracketed by 'space-type' characters - e.g. space, comma, left/right bracket etc. So "-and," "(and)" etc would be located, but "handy" " andrew " " band " would not.

A more useful example of the same principle might be in a Basic program, converting a floating point variable "i" to an integer:

```
ecw/i/i%/
```

will replace "i=i+1" with "i%=i%+1" ; or
"print 'Main items: ';ix(i)" with "print 'Main items: ';ix(i%)".

8.3.6 Find and Exchange - compound form

And there's more ...

The search string may be specified in compound form. That is, more than one search string may be specified. The individual strings may be related by either OR or AND logic. The number of strings that may be specified is user-configurable (see section 8). The program as supplied supports up to three strings.

To find a line which contains an occurrence of "mary", "joy" or "ann", the compound form of Find is needed.

- The command format remains the same, with the same qualifiers. The specification of the search string is however extended. Each string is specified in the usual way - i.e. with delimiters - and between each string is a relationship type (which is either "-" or "+"). The whole lot is surrounded by brackets.

"-" means OR; "+" means AND. The relators may not be mixed in one Find or Exchange.

The name search example is therefore written as:

```
f                        ("mary"-"joy"-"ann").                    or
        fcw(.mary.-/joy/-"ann")       or whatever
```

To change each of these proper names to the string "girlname", the command would be:

```
        ecw ("mary"-"joy"-"ann").girlname.
```

If the "+" relator (AND) is used, then the search is only successful if all of the search strings are found ON THE SAME LINE, and in the correct order. For example, the command

```
        f ("IF"+"END IF")
```

would NOT find the line that contained merely "END IF". It would find a line something like:

```
        "IF condition : do_something : END IF"
```

By exactly the same logic, Exchange using AND related search strings replaces THE WHOLE IMPLIED STRING. If a program contained a series of arrays "day_mon_array$", "day_tue_array$" etc, and these were being amalgamated as "week_array$", then the command

```
        eq(.day.+._array$.)"week_array$"
```

would accomplish the name changes.

8.3.7 Repeated Exchanges

If the Exchange command occurs within a repeat element (an explicit repeat count or the 'RP' command - see 8.14.4), then the Editor will scan across several lines of the file, looking for the search string. When a match is found, if it is not on the current screen page, then the line located is displayed as either the second or the penultimate line on the screen (depending on the direction of search) and the rest of the screen is painted. This screen painting results in a delay. It may be that the user wishes to see the context of the line, in which case the program can be left alone. Conversely, to get the Editor to make the changes as quickly as possible, a few keys should be pressed - this will cause the screen painting to be disabled, so that only the line being changed is displayed. It means that during the repeated exchanges the screen may get a bit messy, but the work is accomplished quicker.

These 'type-ahead' characters are thrown away when an error occurs - as in the case of "Search failed" - which will typically be the termination of a global Exchange, but it is prudent to use RIGHT or LEFT arrow or some other 'non-data' key. The screen display can be entirely disabled by typing ahead 8 or 10 characters.

8.3.8 Special delimiters

There is one additional feature in the operation of string searching. If the search string delimiter is one of the two special characters "<" or ">", then the string match will only be true if the string is located at the beginning ('<') or end ('>') of a line. This feature may be used in any combination with the other features of the Find/Exchange commands.

Examples:

```
  fc>.>                will find (look for) a line ending in full stop
  e<<t; rp<            will insert 't; rp' at the beginning of a line
  fbcw(.IF.+>END IF>)  will find a line containing 'IF' and with the
                                            line ending in 'END IF'
```

8.3.9 Search efficiency

For various reasons, the fastest way of string searching is to use a "case sensitive" search. This requires that the 'C' qualifier is applied to the search. Equally, searching forward is marginally faster than searching backward.

Care should be taken when searching for strings involving 'non display' characters. In such cases, the 'c' - case sensitive - qualifier should be used. Without the 'c' qualifier, the 'zone' bits of each character are ignored, with the effect that for example CTRL/SHIFT/A and CTRL/SHIFT/1 are identified as being the same as each other.

8.4 Insert and Delete commands

8.4.1 Insert prior

```
Command: I                Example: I.now is the time.
 Qualif: Mandatory string
 Effect: Creates a new line prior to the line containing the
         cursor.  The contents of the string become the text of the
         line.  The current left margin is effective - i.e.  the
         line text commences at the defined left margin, and the
         cursor is positioned on the first character of the text.
         If necessary, the "window" pans left, and/or scrolls down.

   Data: Affected
 Errors: Out of memory
```

8.4.2 Insert after

```
Command: A                Example: A.for all good men.
 Qualif: Mandatory string
 Effect: Creates a new line after the line containing the cursor.
         The contents of the string become the text of the line.
         The current left margin is effective - i.e.  the line text
         commences at the defined left margin, and the cursor is
         positioned on the first character of the text.  If
         necessary, the "window" pans left, and/or scrolls up.

   Data: Affected
 Errors: Out of memory
```

8.4.3 Split current line

```
Command: S                Example: S
 Qualif: None
 Effect: Identical to the immediate command "Split current line" -
         see 6.3.2

   Data: Affected
 Errors: Out of memory
```

8.4.4 Delete line

```
Command: D                Example: D
 Qualif: None
 Effect: Identical to the immediate command "Delete line" - see
         6.2.7

   Data: Affected
 Errors: None
```

8.4.5 Delete Character

```
Command: DC              Example: DC
 Qualif: None
 Effect: Identical to the immediate command "Delete character
         right" - see 6.2.2

   Data: Affected
 Errors: None
```

8.4.6 Delete word right

```
Command: DW              Example: DW
 Qualif: None
 Effect: Almost identical to the immediate command "Delete word
         right" - see 6.2.4.  The only difference is that the DW
         command will never allow movement off the current line -
         i.e. the command is ignored if the cursor is beyond the
         logical end of line.  This is for safety reasons, and to
         allow command sequences that clear an indeterminate number
         of characters from the back end of a line.

   Data: Affected
 Errors: None
```

8.4.7 Join line with next

```
Command: J               Example: J
 Qualif: None
 Effect: The line following the line containing the cursor is
         appended to the current line.

         The cursor position does not change.  If the cursor is
         within the logical line, the following line is appended at
         the logical end of line.  If the cursor is to the right of
         the logical line, the following line is appended at the
         cursor position.

         The lower portion of the screen scrolls up by one line.
         The effect is similar to the immediate command "delete
         word right" when the cursor is at (or beyond) logical end
         of line.

   Data: Affected
 Errors: Line too long; End of file
```

8.5 Set position Marker

Command: SM Example: SM
Qualif: None
Effect: The line containing the cursor is noted, and considered to be 'marked'. This line may be recalled/returned to at any time using the 'Cursor to Marker' command CM - see 8.1.14.

The marked line retains its logical identity even though lines may be inserted or deleted, blocks may be moved etc. If the marked line is within the 'block', and the block is moved (see 8.2.4), then the marker moves with the line.

If the line containing the marker is itself deleted, the marker becomes undefined.

Note that there are several other 'markers' with the same kind of functionality. The 'last command point' marker may be accessed with the L command. The start of block may be accessed with the CB command. The end of block may be accessed with the CK command. Each of these may be used as 'location points' within a file. In addition, in for example a SuperBasic program, introducing a blank line - i.e. entirely empty - at strategic points in the file enables the use of the Cursor to Paragraph commands that may also act as effective markers.

Data: Not affected
Errors: None

8.6 Margins and Tabs

Editor provides commands to set 3 margins - the (paragraph) indent margin, the left margin and the right margin. There is also the immediate command that sets a 'temporary' left margin; this has no analogue as an extended command since none is needed.

Two different types of tab structure are allowed. The first and simplest is the 'symmetric' tab, where it is sufficient to set the tabbing interval. The second system is the 'asymmetric' tab structure, where individual tab points may be specified (up to 15 different tab points). These two systems are mutually exclusive.

8.6.1 Set indent margin

```
Command: SI              Example: (1) SI 12   (2) SI
 Qualif: Optional numeric
 Effect: The number is treated as the column number of the indent
         margin - where the first non-space character will appear
         on the first line of a paragraph. It may not exceed the
         maximum length of the line as defined in the Configurator
         (see section 8). Unpredictable results will later occur
         if indent margin stays greater than right margin.

         If no qualifier is supplied, the current physical cursor
         column is used by default.

   Data: Not affected
 Errors: Number expected; Number too big
```

8.6.2 Set left margin

```
Command: SL              Example: (1) SL 8   (2) SL
 Qualif: Optional numeric
 Effect: The number is treated as the column number of the left
         margin - where the first non-space character will appear
         on an inserted line. It may not exceed the maximum length
         of the line as defined in the Configurator (see section
         8). Unpredictable results will occur if left margin stays
         greater than right margin.

         If no qualifier is supplied, the current physical cursor
         column is used by default.

   Data: Not affected
 Errors: Number expected; Number too big
```

8.6.3 Set right margin

Command: SR Example: (1) SR 79 (2) SR
Qualif: Optional numeric
Effect: The number is treated as the column number of the right margin - the column at which word wrap will occur. It may not exceed the maximum length of a line as defined in the Configurator (see section 8). Unpredictable results will later occur if right margin stays less than left margin.

If no qualifier is supplied, the current physical cursor column is used by default.

Data: Not affected
Errors: Number expected; Number too big

8.6.4 Release right margin

Command: MR Example: MR
Qualif: None
Effect: The right margin specification is relaxed for as long as the cursor remains on the current line - i.e. word wrap is temporarily disabled. After the cursor moves off the line, the specified right margin again comes into force.

Data: Not affected
Errors: None

8.6.5 Set Tab interval

Command: ST Example: ST 4
Qualif: Mandatory numeric
Effect: The number is treated as the tab interval definer. If the TAB key is pressed, the cursor will move to the column that is the next integer multiple of the tab interval.

Data: Not affected
Errors: Number expected; Number too big

8.6.6 Tab points Assign Asymmetric

Command: TA Example: TA 5,12,21
Qualif: Mandatory numeric, up to 15 - comma delimited
Effect: Each number is treated as the column number of the tab point required. Any tab points previously defined are ignored. No tab point may exceed the maximum length of the line as defined in the Configurator (see section 8).

Data: Not affected
Errors: Number expected; Number too big

8.6.7 Tab points Remove

```
Command: TR              Example: TR
 Qualif: None
 Effect: Any tab points previously defined are removed. Interval
         tabbing (see ST command) is reinstated.

   Data: Not affected
 Errors: None
```

8.6.8 Tab point Insert

```
Command: TI              Example: (1) TI 17  (2) TI
 Qualif: Optional numeric
 Effect: The number is treated as the column number of the tab
         point required.  The command is ignored if the column is
         already defined as a tab.  No tab point may exceed the
         maximum length of the line as defined in the Configurator
         (see section 8).

         If no qualifier is supplied, the current physical cursor
         column is used by default.

         If interval tabbing is in effect prior to the command, TI
         automatically switches to asymmetric tabbing.

   Data: Not affected
 Errors: Number expected; Number too big
```

8.6.9 Tab point Delete

```
Command: TD              Example: (1) TD 14  (2) TD
 Qualif: Optional numeric
 Effect: The number is treated as the column number of the tab
         point to be deleted.  No problem if the column is not
         marked as a tab.  The number may not exceed the maximum
         length of the line as defined in the Configurator (see
         section 8).

         If no qualifier is supplied, the current physical cursor
         column is used by default.

   Data: Not affected
 Errors: Number expected; Number too big
```

8.6.10 Tabs Compression

```
Command: TC             Example: TC
 Qualif: None
 Effect: The current line is "compressed".  This means that the
         ASCII tab character (value 9 - usually displayed as
         capital I with a bar) - is inserted at suitable places in
         the line to replace a string of successive spaces.  This
         process is controlled by the current tab settings - see
         commands ST, TA and TI.  The cursor is positioned at
         column 1 of the line.

   Data: Affected
 Errors: None
```

8.6.11 Tabs Expansion

```
Command: TE             Example: TE
 Qualif: None
 Effect: Tabs on the current line are "expanded".  This means that
         an occurrence of the ASCII tab character (value 9 -
         usually displayed as capital I with a bar) - is replaced
         by a string of one or more spaces such that the succeeding
         character is positioned at the next higher tab point - if
         any.  This process is controlled by the current tab
         settings - see commands ST, TA and TI.  Any tab characters
         that can not be matched to tab points are left unadjusted.
         The cursor is positioned at column 1 of the line.

   Data: Affected
 Errors: Out of memory
```

8.7 Justification

8.7.1 Set Justify Right mode

Command: JR Example: JR
Qualif: None
Effect: This command causes the Editor to 'right justify' any line of text which is subsequently entered or amended. There are several phases to the logic, and they come into play when the cursor is moved from the entered/amended line. First the line is 'compressed' to reduce strings of successive spaces to only one space (or two spaces following a full stop). The resulting line length is then either capable of fitting within the defined indent/left and right margins or it is not. If the latter, then no further action is taken. Otherwise, the line is 'padded' with spaces such that the last non space character on the line occurs in the column defined as the right margin.

The justify mode affects the operation of 'word wrap' during normal data entry (see 6.3.3).

The justify mode also has an effect on the operation of the 'paragraph reformat' command - see 8.8 below.

Data: Not affected (directly)
Errors: None

8.7.2 Set Justify Left mode

Command: JL Example: JL
Qualif: None
Effect: This command restores the 'default' justification mode. Any line subsequently entered or amended will be left as it is typed (subject to the 'word wrap' action previously described - see 6.3.3). The usual description for this form of line presentation is 'ragged right'.

The justify mode also has an effect on the operation of the 'paragraph reformat' command - see 8.8 below.

Data: Not affected (directly)
Errors: None

8.7.3 Justify Centre

```
Command: JC               Example: JC
 Qualif: None
 Effect: This command differs from the previous two - it does not
         set a mode, but rather is a 'single shot' command the
         effect of which is limited to the current line.

         The leading and trailing spaces on the current line are
         removed and the remaining text is positioned on the
         current line such that the text is centered on the
         mid-point of the line.  The mid-point of the line is taken
         as one half of the right margin width; i.e. for the
         present purpose, the indent/left margins are ignored.

   Data: Affected
 Errors: Out of memory
```

8.7.4 Justify Middle

```
Command: JM               Example: JM
 Qualif: None
 Effect: This command is similar to the JC command.  It has the
         difference that the mid-point of the line is taken as a
         point half way between the left margin in effect - which
         might be the indent, left or temporary left margin - and
         the right margin.

   Data: Affected
 Errors: Out of memory
```

8.8 Paragraph reformat

Command: PR Example: PR
Qualif: None
Effect: When text is being entered, Editor observes the settings of the indent, left and right margins (see 6.3.1-3 etc).

Successive lines of text entered in this way will have the appearance of an organised paragraph, adjusted to conform with the justify mode in effect. If the text is amended subsequently, the format of the paragraph may be corrupted - some lines may be too long, others too short etc. This may easily (and laboriously) be fixed 'manually', but the paragraph reformat command provides a more convenient method.

Editor does not have any real way of knowing what constitutes a paragraph - since it does not and may not put paragraph markers into the file. The rule has been devised that a paragraph is terminated by a blank line or a page break.

The cursor position on the line is important when the paragraph reformat command is given. If the cursor is to the right of the effective left margin, then any text within the left margin is unaffected. If the cursor is within the effective left margin, the character under and those to the right of the cursor are included in the reformatting.

(In the above, 'left' margin should be read as 'indent' margin for the first line of a paragraph, or the temporary left margin - if one is in force).

Reformatting never includes lines 'above' the cursor - even if conceptually they are in the same paragraph. The last line of a paragraph is not right justified, even if right justify is in effect.

Reformatting comprises the same operations that occur when the line is first entered, and therefore is determined by the justification mode in effect when the command is given.

If a full stop is found to be followed by more than one space, then after reformatting there will always be at least two spaces after the full stop.

Margin Release is ignored when a paragraph is reformed.

After a paragraph has been reformed, the screen is repainted. The cursor is usually positioned in column 1 of the blank line following the paragraph.

Data: Affected
Errors: Out of memory

8.9 Case Adjustment

8.9.1 Make lower case

```
Command: ML              Example: ML
 Qualif: None
 Effect: The word under the cursor is set to lower case.  If the
         cursor is not within a word, the command is ignored.

   Data: Affected
 Errors: None
```

8.9.2 Make upper

```
Command: MU              Example: MU
 Qualif: None
 Effect: The word under the cursor is set to upper case.  If the
         cursor is not within a word, the command is ignored.

   Data: Affected
 Errors: None
```

8.9.3 Make mixed

```
Command: MM              Example: MM
 Qualif: None
 Effect: The word under the cursor is set to lower case, and the
         initial character of the word - if a letter - is
         capitalised.  If the cursor is not within a word, the
         command is ignored.

   Data: Affected
 Errors: None
```

8.10 Numbering and Sorting

8.10.1 Renumber the block

Command: RN Example: (1) RN 1000,10 (2) RN ,20
Qualif: Optional one or two numbers, comma separated
Effect: The lines defined as within the block are assigned line numbers, or re-assigned line numbers if line numbers already occur. For the purposes of this command, the block type is assumed to be Line.

The first parameter supplied (which defaults to 100 if not supplied) is used as the line number for the first line of the file. The second parameter supplied (which defaults to 10 if not supplied) is used as the numeric interval between successive line numbers.

It should be noted that this renumbering does not implement a full BASIC RENUMBER - GO TO (etc) commands in the text file are not adjusted. Nonetheless, if the number of lines in the file multiplied by the interval, plus the base number exceeds 32767 the Editor generates an error message and ignores the command.

This facility is provided as a 'block' command so that subsections of the file may be renumbered. Clearly, if required the whole file may be defined as the block.....

If the ESC key is used to interrupt this command, the cursor is moved to the last line that the command had renumbered.

There is no explicit "strip" number command. The effect can be achieved with 'rp n;4dc' or similar.

Data: Affected
Errors: Number too big

8.10.2 Sequence block

```
 Command: SQ              Example (1) SQ    (2) SQ 9,22
                                  (3) SQN 14
  Qualif: Optional 'N'; Optional one or two numbers - "," separated
  Effect: The lines defined as within the block are sequenced in
          ascending order, using the characters between the
          specified columns - the two qualifiers.

          The default values for start and end column are 1 and 80
          respectively.  They may take any value less than the
          maximum line length, and clearly the 'end' column may not
          be less than the 'start' column.

          This facility is provided as a 'block' command so that
          subsections of the file may be sorted.  Clearly, if
          required the whole file may be defined as the block.....

          The 'N' qualifier indicates that the data in the columns
          to be sorted on are to be treated as numbers - perhaps
          with leading spaces.

          If the ESC key is used to interrupt this command, the
          sequence of lines in the block is arbitrary.  The sort
          method used is not a "bubble" sort, so that no sequencing
          of synonyms may be achieved by successive sorts.


    Data: Affected
  Errors: Block not defined; Number too big;
          Syntax error (start column > end column)
          Block in a mess (ESC key used during command)
```

8.11 Memory management

8.11.1 Average Line Length

Command: LL Example: LL 10
Qualif: Mandatory numeric
Effect: When a file is read by Editor, the program is able to establish the total size of the file before reading. An appropriate amount of data area can be requested to accomodate the file data.

Editor additionally requires to maintain a control area for the file, and the size of this control area is a function of the number of lines in the file. This Editor can not know until the file has been read.

Depending on the file type (text or unformatted), Editor "guesses" at the average line length. It is usually not too important that this guess should be that accurate. However, if a large file contains a high proportion of blank lines or if almost all of the lines are very short, then the guess will be inadequate. It is possible that an "Out of Memory" error will be displayed on an attempt to read such a file.

One way round is to allocate much more memory than the file actually needs. This may be either inefficient or impossible. The option is to tell Editor of the abnormal average line length. This will then allow the program to compute an optimally sized buffer requirement.

Normally, after processing the non-standard file, the average length should be reset to 40.

Data: Not affected
Errors: Number expected; Number too big

8. Specify new memory size

```
   Command: M                  Example: M 440
    Qualif: Mandatory numeric
    Effect: The numeric qualifier is treated as a number of "K" to be
            requested from the operating system.  Editor checks to see
            that that amount of space is potentially available.  If it
            is not, an error message is displayed.

            If the current file in memory has been altered since it
            was last saved (or read), Editor requests confirmation of
            the command.

            If the command proceeds, the whole of the file in memory
            is deleted, and the current memory area released to the
            system.  Editor then puts in a request for the specified
            memory space.  All file control information in the program
            is reset.

            Note that this command may be used to REDUCE as well as
            INCREASE the amount of memory retained by Editor.

            Usually, if this command fails, the current file contents
            (if any) remain available.  There are circumstances - due
            to fragmentation of QL memory space by (say) definition
            blocks or RAM discs - where the current file data has been
            released because the memory requested was seen to be
            available IN TOTAL, but the command will still fail
            because it transpires that the free memory is not all in
            one chunk.

      Data: Affected
    Errors: Out of memory
```

8.11.1 Zap the current (memory) file

```
   Command: Z                  Example: Z
    Qualif: None
    Effect: The whole of the file in memory is deleted.  If any
            changes have been made to the file since the last save,
            Editor requests confirmation of this command.

      Data: Affected
    Errors: None
```

8.12 Undo

8.12.1 Undo changes

```
Command: U                Example: U
 Qualif: None
 Effect: The changes made to the current line are removed.  This
         command is effective for as long as the cursor is not
         moved from a line which has been changed.  Once the cursor
         is moved off the line, the changes are committed to
         memory.  The command may not be used to recall a deleted
         line or block.
```

> Note that the Show Status command SH effectively disables the Undo command, since the cursor is considered to have moved off the line - even though it has not.

```
   Data: Affected
 Errors: None
```

8.12.2 Undo Deleted line

```
Command: UD               Example: UD
 Qualif: None
 Effect: The text of the last deleted line (if any) is inserted as
         a new line PRIOR to the line currently containing the
         cursor.

         No history of line deletion is retained - only the
         immediately previous deleted line.  The Undo deleted
         command may be issued as often as is required.  This
         feature may occasionally serve, therefore, as a second
         type of "block" handling capability.

   Data: Affected
 Errors: Out of memory
```

8.13 Command files

A command file is, as its name implies, not much more than a file of Editor commands. The facility for command files enables several objectives to be met.

On occasion, it is necessary to perform a series of similar "one-shot" operations - e.g. a global replace for "this" to "that" - where there are several values for "this" and "that". It is often convenient to prepare all of the commands at one time - perhaps using editing commands to build the commands - and then execute them all at one go. This latter is perhaps simplest done by writing the commands to be executed to, say, a RAM disc file.

There are other techniques available in Editor (e.g. the EX command) that may suit just as well, but that is not the point here.

On a different tack, it may be that a complex series of commands is required to achieve a certain result, and that operation is going to be required on a regular or frequent basis. Then it makes sense to stack up the command sequence in a command file, rather than (a) having to sit over Editor while it is executing the commands and (b) running the risk of "finger trouble" while typing the commands.

In this second case, it may be difficult or impossible to try to write commands that have the full flexibility required. A further feature of the command file processor may help out.

The RC command supports up to 9 parameters, so the command form is:

```
        rc.file_name. \n'para n' \m'para m' etc
```

For example:

```
    rc.fred_cmd. \1 '28'  \2 '5'  \3 'work_file'
    rc.mary_cmd. \6 ?apples?  \4 :2.37:   \9 ""
```

When the sequence \n is encountered in a line in the command file Editor makes the appropriate substitution (so in the first example, \1 everywhere is replaced by 28 and in the second example \6 is replaced by the string apples). If a single backslash is required in the command file, the sequence \\ should be used.

The command format looks a bit fearsome, and has the further implication that you have to remember what parameters are needed and what the parameter numbers are. The idea of parameters has been taken a little further, so that you may put commands into command files that prompt for their own parameters - in English. This means that there is no need - unless you want to - to provide the parameters within the RC command itself.

As a further aid to keeping track of what is going on, you may put comments into command files. These are lines that begin (column 1) with the full stop character.

Some simple command files are provided on the distribution volume. We tend to use the suffix "_cmd" for Editor command files, but that is purely a matter of choice. The file names are "demo_cmd", "column_cmd" and "quill_cmd".

If a command file contains a line that is syntactically incorrect, or if any other type of error occurs during command file processing, Editor will stop command file execution at that point. The command file is closed and the error message "Syntax Error" is displayed.

If the operation of the command file was unattended, i.e. no one was watching what was happening, it may be a problem to discover which line was the culprit.

In such cases, Editor displays a "line number" in the error message. The line number is always negative, to reinforce the idea that a command file is the origin of the error. (Positive numbers may/will be produced by the "rp" command in direct mode).

The line number is the physical line in the command file, starting with line 1, and includes comment lines (which are lines beginning with a dot/period/full stop).

8.13.1 Read command file

Command: RC Example: RC .mdv2_partprint_cmd. ..params..
Qualif: Mandatory string; optional parameters
Effect: An attempt is made to open the named file on the named device. If the file exists, successive lines of the file are read and treated as though they were command text.

One feature distinguishes commands coming from a file, and that is that navigation errors are not treated as fatal - they merely terminate the current command line.

For example, if the command "rp e.fred.joe." were entered from the keyboard, at end of file, the 'error' message "Search failed" would result. With the same command coming from a file, the command terminates and the next line of the command file (if any) is then processed. Similarly for the error messages "End of file", "Top of file" and "Block not defined".

Any other error conditions (including pressing ESC) cause the processing of the command file to cease. The 'read command file' command may not be contained in a command file.

As each command line is read from the command file, it is displayed in the command window in the usual way, until / unless displaced by the next command or the results of the command in effect.

The command file remains open for the duration of the "rc" command, and the media in the device containing the command file should not be removed until the command has ended.

If an error occurs while processing a command file - other than those listed above - the error message that is displayed will show the reason for error in the usual way, but will also display a NEGATIVE number. The minus sign should be ignored - it is there simply to distinguish the number from that produced at the end of an RP command. The absolute value of the number indicates the line number of the line in the command file that caused/contained the error.

Data: Not affected (directly)
Errors: File not found; Command file open; Commands abandoned

8.13.2 Ask for Parameter

Command: AP   **Example:** AP2 "Enter the file name: "

Qualif: Mandatory numeric (1-9); Optional string

Effect: This command is exceptional in that it may ONLY be executed from within a command file.

The purpose of the command is to request - from the command file user - a value to be assigned to the "nth" parameter of the command file. As indicated previously, a command file may have up to 9 parameters.

The numeric qualifier indicates which of the parameters is to be assigned the incoming value.

The string qualifier contains the message that will be displayed to the user when the command is executed. If the string is not supplied, the message text will default to "Enter parameter N: " - where N will be the appropriate number.

The response from the user is not checked in any way for validity. It is simply stored as the current value for parameter N, to be used as a substitute when a subsequent line in the command file contains the character pair "\N". From this it can be seen that, if necessary, a command file may contain several AP1 or AP2 etc commands.

Data: Not affected

Errors: String too long

8.14 Miscellaneous commands

8.14.1 Sound Buzzer

Command: BZ Example: BZ
Qualif: None
Effect: The command causes the QL buzzer to emit a standard sort of wail.

The purpose of the command is principally to signal the end of processing of a long command file.

Mildly interesting effects can be achieved by various command combinations, and these differences may be used to denote different stages of progress through a command file. For example:

```
12(3bz bz 2bz)
bz 50bz bz
rp bz
```

This last example of course needs to be at the end of a command sequence. It can be 'broken' by ESC, as usual.

Data: Not affected
Errors: None

8.14.2 Set Cursor Delay

Command: CD Example: CD22
Qualif: Mandatory numeric
Effect: Sets a delay for cursor repeat - from 1 to 30000. The objective of this command is to allow the user to tailor the movement characteristics of the cursor.

The problem is that with such a variety of memory hardware around, with widely different performance characteristics, it is close to impossible to tune Editor to provide very fast cursor movement AND immediate stop AND constant visibility of the cursor.

People with fast RAM - CST for example - will probably find that a CD of about 180 is satisfactory. Experiment with different CD settings on your own installation, and you will find a setting that you can be happy with. Note that this figure may be established in Editor once and for all, using the Configurator - see section 11.

Data: Not affected
Errors: Number too big, Number expected

8.14.3 Execute current line

Command: EX Example: EX
Qualif: None
Effect: The command causes the line containing the cursor to be executed as though it were a command line (hopefully it is one).

Note that column 1 of the line is (presently) ignored. This is to allow later releases to be able automatically to recognise embedded commands in an (otherwise) data file. At that time, if the first character on the line is semi-colon, the line will be treated as an embedded command. For the time being column 1 may contain anything - it is not examined.

This feature allows you to record various types of otherwise transitory information in the file, for example a data file may start with the following:

```
  ; sl12 sl7 sr68;  n ex
  ; ta 7, 12, 22, 43; btc; bh
```

A line which is executed MAY NOT contain the EX command, unless the cursor has first been moved to a different line. Even then, loops are not supported. This is in your interests. The sequence

```
  ; -----  n ex
  ; ---------- p ex
```

is non productive and possibly dangerous.

The line which is EXecuted may contain the RP command, and the EX command may itself be within an RP group. In this way it is possible to establish loops within loops. These comments of course refer to commands contained in a command file - otherwise the error which terminates the inner RP is not suppressed (see section 8.14.4 and 8.13).

Data: Not affected
Errors: None

8.14.4 Repeat

Command: RP Example: RP ewq/THEN/:/

Qualif: Always followed by one or more commands

Effect: The Repeat command causes all following commands on the command line to be continuously executed until some "error" condition occurs. The "error" will usually be that a search string is not found, or that top or end of file has been detected, or that the user has pressed the ESC key. Commands following the Repeat are executed in sequence. When the last command on the line has been processed, then the first command after the RP is again executed etc.

There is logically no sense in having RP commands nested on the same command line, but finite repeat counts may occur within an RP group. However, if operating from a command file, the EX command may be contained within an RP group, and another RP command may meaningfully be contained in the line which is the subject of EX. In this way, nested repeats are possible and useful.

When the RP command is given 'directly' - i.e. not in a command file - it always terminates with an exception message - something like "Search failed" or "End of file" etc. This message includes a number reporting the number of times that the repeated group of commands has been successfully executed - e.g. "Search failed 13".

A command such as "t rp f/fred/" does not appear to do much other than place the cursor on the last occurrence in the file of the string "fred". However, the command does actually have the benefit that it reports how many occurrences of the string "fred" have been detected.

It is the user's responsibility to ensure that the repeated commands are sensible. Examples of fairly pointless commands are those that do no effective work, and will therefore not result in an exception condition. Consequently, the command will not stop.

```
(1)        RP
(2)        RP   CR   CL
```

Commands of this type can only be halted using the ESC key.

Data: May be affected by the repeated commands

Errors: None due to the RP command

8.14.5 Show status

Command: SH Example: SH
Qualif: None
Effect: A window is opened on or near the main screen, and the status of various system parameters is displayed. The user is required to press a key before the main edit screen is restored. The variables displayed are as follows:

Current file name and type (and page length/word count)
Left margin, Indent margin, Right margin
Tabbing interval OR Tab points
Justify mode, Block type
Block start, Block end (line number and some of the text)
Marker point (line number and some of the text)
Last Find or Exchange command
Last replacement string
Current data buffer size and percentage usage
Current control buffer size and percentage usage

Data: The current line is stored before info display. The 'Undo' command is therefore ineffective after the SH command. Otherwise data not affected
Errors: None

8.14.6 Make Document file

Command: MD Example: MD

Qualif: None

Effect: The effect of this command is to scan the whole of the current file in memory, converting the file form from a text file to a document file. This involves tallying the number of 'words' on file, computing page breaks etc.

The page length used for computing page breaks is determined by the value last set by the PL command. The definition of a word is based on the user's specification for the Find/Exchange word delimiters.

The command will take an amount of time proportional to the length of the file. A guideline figure is about 10,000 words per minute. Any subsequent MD command for the same file is ignored, since Editor will automatically maintain the word count and page control for the file through all of the subsequent editing operations. This may result in a slight increase in response time. In a very lengthy file, response time will be badly degraded for line insert/delete if there are few/no 'hard' page breaks.

The word count may be inspected via the SH command.

The command is not allowed on a file that was read with the RU (read unformatted) command.

Reclassifying the file as a "document" file - as opposed to a normal text file - has certain consequences. In particular, Editor will regard "non-display" characters as print highlighting characters - e.g. boldface, underline or whatever - and will not therefore consider these characters as "occupying space". The visible effect of this is that the column indicator will not be incremented as the cursor is moved across these characters. Additionally, when word wrapping or reforming paragraphs Editor works to the "logical" length of the line - i.e. the effective length in the absence of the control characters. This has the effect that 'right justified' paragraphs may appear on screen to be 'ragged right', but will appear correctly when printed.

A further effect is that any line starting with ";" - semicolon - in column 1 is regarded as a "non-print" line. It will be retained in the file, but will not be included in the line count for paging. Nor will the line be sent to the printer when using the WP command. These lines are intended to contain formatting, heading, footing etc information. They may be edited as normal, and will be included in the word count of the file.

Data: Not affected

Errors: Syntax error (file type is unformatted

8.14.7 Set Command screen colours

Command: KC Example: (1) KC 1, 4 (2) KC ,6
Qualif: Optional numeric pair (range 1 to 255)
Effect: The first number (if any) is used as the new ink colour to be used for the command/status window.

The second number (if any) is used as the new paper colour to be used for the command/status window.

Data: Not affected
Errors: Number too big

8.14.8 Set Error screen colours

Command: KE Example: (1) KE 1, 4 (2) KE ,6
Qualif: Optional numeric pair (range 1 to 255)
Effect: The first number (if any) is used as the new ink colour to be used for the error window.

The second number (if any) is used as the new paper colour to be used for the error window.

Data: Not affected
Errors: Number too big

8.14.9 Set Main screen colours

Command: KM Example: (1) KM 2, 1 (2) KM 3
Qualif: Optional numeric pair (range 1 to 255)
Effect: The first number (if any) is used as the new ink colour to be used for the main window.

The second number (if any) is used as the new paper colour to be used for the main window.

Data: Not affected
Errors: Number too big

8.14.10 Set document Page Length

Command: PL Example: PL 54
Qualif: Mandatory numeric (range 6 to 255)
Effect: The command allows the specification of the page length to be used by Editor to compute "soft" page breaks. Soft breaks occur because the number of lines on the page exceeds the specified length.

A "hard" page break may be specified at any time by entering the "form feed" character (ASCII 12 - CTRL/L) as the first character of any line (column 1).

The command may be used with any file type, but only has an effect with a document file. It may be specified before using the RD or MD command.

When a document file is being processed, the occurrence of a page break is (usually) denoted on screen by a hatched line. This is a screen convention. No additional data is included in the file by Editor to record a 'soft' break.

Data: Not affected
Errors: Number too big

8.14.11 Hide Paging

Command: PH Example: PH
Qualif: None
Effect: The command allows the display of page breaks for a document file to be switched on or off.

When page breaks are displayed by Editor, the user may notice an increase in the response time for some commands. If there is a fair amount of 'structural' reorganisation (inserting, deleting and moving lines or groups of lines) required on a file, the display of page breaks during the editing may be irrelevant, and the editing work done quicker with the page control switched off. Hard page breaks improve response time. Soft page breaks don't.

The page break display may be turned off, the work done, and then the break display turned on again.

There is a small time penalty when turning the breaks off and when turning the breaks on, dependant on the size of the file.

Note that even though page break display is disabled, the file is still being processed as a 'document' - i.e. the word count is maintained and the treatment of 'non-display' characters is consistent with a document.

Data: Not affected
Errors: Syntax error (not a document file)

8.15 File commands

In all of the file commands, where ever there is a requirement to specify a file name, the file name may be entered without a device identifier. Editor will use the 'default data device' as a prefix. The default device identifer is specified during the Configurator dialogue (see 11.18).

Almost all of the file commands also have the capability to access devices - for example the network, serial and parallel ports of the QL.

Every effort has been made to 'protect' program operation from device related errors. Checks are made before access to any device or file to ensure that everything is in order before any access command is implemented. Nonetheless, it may be possible for error conditions to occur during file operations - the infamous "bad or changed medium", removal of a disc or tape during read and write operations etc. These errors also are trapped by Editor, but not as gracefully as error conditions that may be detected before the operation commences.

When a system error occurs which is outside of the normal scope of Editor operations - and the classic if not only example is device related errors - the Editor screen is cleared and a message appears which contains the QL standard error code appropriate to the fault. Control may be returned to the main Editor process by pressing any key. The current file in memory (if any) will be re-displayed, with the cursor repositioned to the top of file.

Some examples of the QL standard error codes (a full list of which is contained in all standard QL publications) are:

-3	Out of memory
-4	Out of range
-9	File or device in use
-11	Drive (disc or tape volume) full
-16	Bad or changed medium

8.15.1 Read (text) file

Command: R Example: R.flp2_archive_prg.

Qualif: Mandatory string

Effect: If any changes have been made to the file in memory since it was last saved, the Editor requests confirmation of the command. If confirmation is received, the command continues, otherwise the command, and any following, is abandoned. The named file is sought on the indicated device. If the file is found, the length of file is determined. If the memory already acquired by the Editor is sufficient, the file is read. Otherwise, acquired memory is released and the Editor obtains a new allocation, based on the length of the file and about 15% expansion. All trace of the previous file (in memory) is removed. The screen is refreshed with the new file contents and the cursor is placed at line 1 column 1. Left and right margins, tab points/interval, search and replacement strings are unchanged. The block is undefined. The marker is undefined. The file class is 'text'.

The memory management just described is intended as a convenience to the user. There may be situations when it is known beforehand that a small file is going to have several other files appended. In this case, the amount of memory required is unrelated to the size of the first file read. The solution is to specify the buffer size required - using the M command - before reading the first file. The amount of memory specified should be large enough to contain all of the required files.

When a file is read using this command, it is expected that the file contents are formatted as a text file - i.e. displayable characters terminated by a linefeed (LF) character. Lines that are longer than the defined maximum are truncated. On input and during file editing, trailing spaces on a line are deleted.

The ESC key is normally disabled during this command, since no useful purpose is served by interrupting it. The exception is for 'non directory' devices (serial port / network), where a comms breakdown may have occurred. In such cases pressing the FSC key will terminate the read file command.

The file remains open only for the duration of the command. The media (tape, disc or whatever) in the device may be removed when the command is ended.

Data: Affected

Errors: File not found; Out of memory; Lines truncated

Message: Reading: ..file name..

8.15.2 Read (unformatted) file

Command: RU Example: RU.flp2_edt_bin.

Qualif: Mandatory string

Effect: The general activity of this command is the same as that described for the Read text file command.

The principal difference is that the Editor does not expect a neatly constructed text file, and is prepared to handle whatever materialises.

Input lines that do have a Linefeed terminator will be displayed as normal. Input strings that are in excess of the maximum line length are split at the maximum length, and the continuation occurs on the following screen line.

On input, therefore, lines will not be truncated. Neither will trailing spaces be deleted on input or during editing.

The data entry mode is reset to "overstrike", and the right margin is set to the maximum line length.

The Editor contains no other 'syntax' or file type related checks, so that the user may make whatever changes s/he chooses to the edit file. The sense or nonsense of the result is the user's responsibility. For example a binary file (object program) may be edited such that one string (say 'mdv1_') is changed for another (e.g. 'flp1_'). This is likely to be ok. It is however unlikely that 'mdv1_' could be changed to 'the quick brown fox' without resulting in some serious faults in the subsequent operation of the edited program. Notwithstanding, the Editor will happily do it if told to.

There is a minor addition to the Status line, when processing an unformatted file. The decimal value of the character under the cursor is displayed, after the data entry mode item.

Data: Affected

Errors: File not found; Out of memory

Message: Reading: ..file name..

8.15.3 Read document file

```
Command: RD                Example: RD.flp2_report6_edt
 Qualif: Mandatory string
 Effect: Initially, this command is identical to the Read text file
         command.  All comments in section 8.15.1 apply here.

         In addition, during the file reading process, the word
         count and page structure of the file are established. The
         effect is functionally similar to the R command followed
         by the MD (Make Document) command.

         All subsequent processing of the file is to 'document
         standards' - namely page control is maintained, word
         counts are maintained, 'non-display' characters are
         regarded as occupying no space at print time, and
         'non-print' lines are not included in the page line count.
         More detailed comments are given in section 8.14.6.

   Data: Affected
 Errors: File not found; Out of memory; Lines truncated
Message: Reading: ..file name..
```

8.15.4 Append file

```
Command: AF                Example: AF.mdv1_sortproc_bas.
 Qualif: Mandatory string
 Effect: The named file is sought on the indicated device.  If the
         file is found, the length of file is determined.  If the
         memory already acquired by the Editor is sufficient, the
         file is read.  Otherwise, the command is abandoned.  The
         incoming lines from the file are added to the text file in
         memory , commencing at the line below the line containing
         the cursor.  The file format of the appended file is
         assumed to be the same as the file in memory (see 8.15.1,
         8.15.2 and 8.15.3), and the incoming lines are treated
         accordingly.

         Editor does not retain the device after this command is
         complete.

   Data: Affected
 Errors: File not found; Out of memory; Lines truncated
Message: Reading: ..file name..
```

8.15.5 Write file

Command: W Example: (1)W (2)W.mdv2_newtext.
Qualif: Optional string
Effect: If a file name has been previously established (via the "read file" or "write replace" commands), then the filename string is optional. Otherwise, the filename string is mandatory. An attempt is made to open the named or implied file on the named or implied device. If the file exists, the Editor asks for confirmation to overwrite. If confirmation is not given the command and any following is abandoned. The Editor computes the size of the output file and checks that sufficient space remains on the device (allowing for removal of the current file if any). If this test fails, the command is aborted. The final check ensures that the named device is not "read only". NOTE THAT THIS CHECK DOES NOT WORK FOR MICRODRIVES - this is a QL fault. Having satisfactorily completed these checks, the output file is opened and the text file in memory is written.

If the file originally read (and now being written) is known (by Editor) to be an object program, then a suitable file header will be written. On the QL, an executable file has a slightly different header from a normal text file. In effect, an SEXEC command will be used to save the file.

ESC during this command results in a confirmation request. If the response is 'y' or 'Y' (or another ESC), the write file command is abandoned.

The above description is phrased in terms of 'filename'. In fact, the string may identify a device - e.g. "ser1" or "net6" etc - so that the command may be used for example to print a file.

Editor does not retain the file or device after this command is complete.

Data: Not affected
Errors: Can't open file; Make space on device; Commands abandoned
Message: Writing: ..file name..

8.15.6 Write replace

```
Command: WR               Example: WR.mdv2_newtext.
 Qualif: Mandatory string
 Effect: This command is identical to the "write file" command,
         with the addition that the file name specified replaces
         the internal default file name, and that the 'file name'
         must be a file name - it may not be a device.  This
         command is useful when a new file is created in memory and
         is written to disc/tape for the first time.  Subsequent
         "write file" commands may then use the default form 'W'.

         Editor does not retain the device after this command is
         complete.

   Data: Not affected
 Errors: Text  expected; Can't  open  file;  Make space  on  device
Message: Writing: ..file name..
```

8.15.7 Write block

```
Command: BW               Example: BW.mdv2_module.
 Qualif: Mandatory string
 Effect: An attempt is made to open the named file on the named
         device.  If the file exists, the Editor asks for
         confirmation to overwrite.  If confirmation is not given
         the command and any following is abandoned.  The Editor
         computes the size of the output file and checks that
         sufficient space remains on the device (allowing for
         removal of the current file if any).  If this test fails,
         the command is aborted.  The final check ensures that the
         named device is not "read only".  NOTE THAT THIS CHECK
         DOES NOT WORK FOR MICRODRIVES - this is a QL fault.
         Having satisfactorily completed these checks, the output
         file is opened and the defined block in memory is written.

         As with the other 'write' commands, the write block
         command supports a device as the destination, so that
         parts of a file in memory may be, for example, printed.

         ESC during this command results in a confirmation request.
         If the response is 'y' or 'Y' (or another ESC), the write
         file command is abandoned.

         Editor does not retain the device after this command is
         complete.

   Data: Not affected
 Errors: Block not defined; Can't open file; Make space on device
         Commands abandoned
Message: Writing: ..file name..
```

8.15.8 Write to printer

Command: WP Example: WP.ser1. 6,9
Qualif: Mandatory string; Optional numeric pair
Effect: This command is provided to allow 'draft' prints from a "document" file in memory.

The numbers optionally following the device specifier are used as 'start page' and 'end page' identifiers. The applied defaults are '1' and 'end', should either number not be supplied.

The data written to the printer is slightly reformatted in the following ways:

1) A page throw occurs at each page break
2) 'non print lines' - those with ";" in column 1 - are not printed.

> It is expected that future versions of Editor may extend the scope of this command to integrate some or all of the functionality presently provided in "edtprt_bin".
>
> This includes headers, footers and automatic replacement of edit-time conventional codes with the appropriate printer control codes.
>
> For the present, to obtain these features, the file should be printed external to Editor, using "edtprt".

Editor does not retain the device after this command is complete.

Data: Not affected
Errors: Text expected; Can't open file; Make space on device
Message: Writing: ..file name..

8.16 Terminate program commands

8.16.1 Quit

```
Command: Q                Example: Q
 Qualif: None
 Effect: If any changes have been made to the file in memory since
         it was last saved, the Editor requests confirmation of the
         command.  If confirmation is received, the command
         continues, otherwise the command is abandoned.  The
         program terminates.

   Data: Not affected
 Errors: None
```

8.16.2 Quit with save

```
Command: X                Example: X
 Qualif: None
 Effect: This command is equivalent to the command 'W; Q'.  The
         program terminates after writing the text file.

   Data: Not affected
 Errors: None
```

9. Error and warning messages

Error messages and warnings are displayed in the position of the Status line, using a characteristic ink and paper colour. In the description of immediate and extended commands, reference was made to the sort of error conditions that may occur with each command. Those references were merely indications. The full list of possible error messages is as follows:

```
"Bad file name" - device type missing or name > 33 characters long
"Block not defined" - Block command with no block defined
"Block in a mess" - Block sequence command interrupted with ESC
"Can't open file" - Output file can't be opened (?read only?)
"Commands abandoned" - ESC pressed or negative response to Confirm
"Command file open" - The 'RC' command detected in a command file
"Cursor inside block" - Block move or insert to itself
"End of file" - cursor movement attempt beyond last line
"File not found" - Input file not located
"Lines truncated" - Text file contained lines longer than max
"Line too long" - Current line plus the req'd addition exceeds max
"Make space on device" - Output file won't fit onto device
"Marker not defined" - Cursor to Marker command with no marker
"Number too big" - Supplied number out of permitted range
"Number expected" - Command needs a trailing number (e.g. ST,PL..)
"Out of memory" - Out of memory condition
"Search failed" - Search string not matched in the search area
"Syntax error" - A mistake (?delimiter missing?) on command line
"Text expected" - Command needs a trailing string (R,WR,AF,I,A)
"Text too long" - Search/replace string etc > defined max length
"Too many tabs" - The limit of 15 tab points has been exceeded
"Top of file" - cursor movement attempt prior to line 1
"Unknown command" - Invalid 'immediate' command or key stroke
```

If the message occurs from a command contained within an RP (repeated) group, the message will be followed by a POSITIVE number - zero and upward - showing the NUMBER OF TIMES that the group has been repeated. RP reporting of this kind is suppressed when the commands are being executed from a command file.

If an error occurs while processing a command file, the error message will be followed by a NEGATIVE number. This number indicates the LINE NUMBER of the line in the command file that caused the error.

Warning messages typically occur in three situations: Confirmation of text replacement, Confirmation of file read or program terminate over an amended memory file, and Confirmation of a file overwrite.

To the first two, the text is "Type Y to confirm: ". To the third, the message text is "Overwrite ..filename.. (y/n): "

One other type of message may occur in the prompt message area. This is the text of the message supplied by the AP (Ask Parameter) command optionally used when processing command files.

10. Some sessions with Editor

This section is intended to give an understanding of how some of the commands may be used. The example situations are more or less genuine - although they may not apply to every user - but the point of the section is the way the commands are being used, not the actual 'problem' being solved.

10.1 Playing with directories

Assume that you need to copy a lot of files from a disc containing many files, and there is no simple lexical rule for the files you do or do not wish to copy. The answer is to do each one 'by hand' (i.e. the COPY or WCOPY command) or use Editor. Let's use Editor.

The disc to be copied to is in flp2_. The disc to copy from is in flp1_. Create a directory of flp1_ in a file on flp2_, using the standard SuperBASIC commands as follows:

```
open_new #3,flp2_direct : dir #3, flp1_ : close #3
```

Fire up Editor (or CTRL/C if it's in memory), and read the file you have just built. This file consists of two heading lines, followed by all of the file names - one name per line - of the files on flp1_.

```
r.flp2_direct.
```

Get rid of the two heading lines and turn each of the other lines into a copy statement of the form "copy flp1_xxx to flp2_xxx"

```
2d; btl rp bs;be;bl;  e..copy flp1_.;  e>> to flp2_>;  j; n
```

That's that done. Scan through the edited file and delete the lines corresponding to the files you don't want copied. Write the file back, get out of Editor, LRUN flp2_direct and go make some tea while it's running. If you like prettily organised discs, you could even sort the file names into alpha sequence before you did the copies.

You can play similar games with wdir, wstat etc - if you've got those commands.

Generally, for lots of different jobs of this kind, Editor will save you stacks of time. All it takes is the mental attitude "how can I get Editor to do it for me" - there's a great chance that Editor can.

10.2 Creating an Index

Assume that the main body of this document has been prepared in a file called 'edt_prt'. The present objective is to prepare an Index. Some parts of the Index preparation are intuitive, but quite a lot is mechanical. We will get Editor to construct the bulk of the Index.

```
r.flp1_edt_prt
```

This command reads the file, having acquired sufficient memory space to allow the file to be loaded, with a margin for expansion.

What is needed is to pick out all the sub-headings. These are characterised by two full stops on the same line. There will be other lines that also have two full stops, but that can be sorted later. The scheme is to collect all these lines at the back end of the file, then throw the main file away and concentrate on getting the index right.

```
b; sm; t; 150 ( f(/./+/./); bs; be; b; bi; cb; ce )
```

Ignore the bit before 150 for the moment. The find command gets the first subheading line (or something that might be one). That line is defined as the 'block', using bs be. The cursor is moved to the bottom line of the file, and an image of the block is copied there. The cursor is then moved back to where we came from (either cb or ck will do just as well). To avoid finding the same full stops on the same line, the cursor is moved to the end of line.

The repeat count of 150 on the group causes the whole command group (the bit within the brackets) to be repeated. The find command at the front ensures that we actually move through the file, finding successive occurrences of the search pattern.

It would have been wrong to use an RP here, since eventually we will begin to find the lines that have been copied at the end of file, and the command would then only stop when memory was exhausted. So we guess at 150. Another more precise way would be:

```
b; sm; ce; rp  fb(/./+/./); bs; be; b; bi; cb
```

geddit?? - it's the same thing only working from the end of file to the top.

Either way, after the command is over, the back end of the file contains all the lines that we want and some that maybe we don't. First off, the main body of the file can be thrown away. Position the cursor on the last line of the proper file, define the file as the block, and delete the block:

```
cm; be; t; bs; bd; wr.flp1_edt_idx
```

In both of the above methods, the first action was to mark the last line of the file proper. The cm command returns the cursor to that point. The wr ensures that we don't inadvertently overwrite edt_prt later on. Visual inspection of the remaining lines is required to delete 'non-index' lines. All the lines left are then of a standard form - being the sub section reference in columns 1 to 7 and the subsection title in column 8 onwards.

Any other 'intuitive' lines should now be added, keeping the new entries to the same format. Some of the existing lines may require to be duplicated (probably using bs;be;bi), followed by a slight modification to the section heading so that an entry occurs as Block - mark start and Mark start of block etc.

```
t; bs; b; be; sq 8
```

Having got all the basic entries in the file, with one entry per line, the file can then be sorted on the section heading, or more accurately now the index entry name.

Each line is the wrong way round. We want the entry name first, followed by the section reference.

```
t; 1//; rp n; 7cr; s; p; bs; be; n; bm; 44cr; j
```

The solution is to split each line just before the entry name, move the section reference from what is now the line above to the line below and then join the two lines, allowing for a variation in length of entry names.

The final stage of editing is manual - inspect for duplicate entry names and join the lines together, and any other adjustments according to one's taste in index layout. Save the file, re-read the main document and append the index.

```
w; r.flp1_edt_prt.; b; af.flp1_edt_idx.; x
```

10.3 Creating a command file

To create a command file is simplicity itself. Once you have decided what you want the command file to do, simply write it!

For some reason, it is decided to write a command file to delete the current document (ie; to clear Editor's memory). From within Editor, press F3 (command mode) & enter:

```
BTL T BS B BE BD
```

which clears any file you may already have in Editor. Now type, on the first line on the screen, the same characters

```
BTL; T; BS; B; BE; BD
```

followed by Enter. (Note that the use of semicolon and space is a matter of taste). Then press the function key F3 and enter:

```
W.mdv2_zap_cmd
```

which will save the line of text as a file called "zap_cmd" on microdrive 2. Now read in a file - if you don't have one handy, use one of the files on the Editor tape itself - say EDT_HELP. Once the file is loaded, enter the command:

```
RC.mdv2_zap_cmd
```

which will execute the command file called zap_cmd. Hey presto! The file you had loaded in has been zapped. Just as well building the command file didn't take too long - the Z command does exactly the same job.

NOTE: Particularly for us lazy types. In the two file commands illustrated above (W and RC), you'll note that the terminating delimiter was not specified - i.e. there was no dot after the file name. This is an example of Editor being kind. If a delimited string is the last thing on a command line, Editor does not insist that the terminating delimiter is typed. It is very easy to get into the habit of not using them.

You can however get caught out if you completely forget that they are supposed to be there. For example:

```
W.fred R.mary
```

will not do what you want. It should more correctly be:

```
W.fred. R.mary
```

10.4 Initialisation

Another use for command files is in initialisation. One of Editor's great strengths is that it stores all files in a 'pure' form - ie; if you have a file comprising:

```
HELLO
GOODBYE
```

on consecutive lines, with the cursor on the first column of line 3, the file will comprise exactly 5+1+7+1+1=15 characters when it is saved to a device (the 12 visible characters, two Linefeed characters - one at the end of each line, and an End of File character).

From this you can deduce that Editor does not store data re left/right/indent margins, justification etc with the document. This is not to say that the reloaded document will look any different from when it was saved - it won't. The only consequence is that subsequent editing of the saved document will not take into account the margins/justification settings at the time the document was saved. Command files hence provide a very useful way of restoring default settings each time you load Editor.

If, for example, you prefer a setting of left margin to 5, right margin to 65, indent to 7 and right justification ON for most of your document work, save a file comprising

```
BTC BH   SL 5; SR 65; SI 7; JR
```

with filename (say) mdv1_init_cmd. Now whenever you wish to restore the settings in Editor to these values, enter the command:

```
RC.mdv1_init_cmd
```

Note that much of this can be done by using the Configurator anyway. Another thought on the same lines is that specific margin or tab settings that you have set up CAN be remembered while a document is being developed. You can write a command line into your document file. For example:

```
; ta 12,18,23,33,45,57  si 8 sl 8 sr 74 jr
```

When the file is next read in, simply place the cursor somewhere on this line and use the EX command. Of course, you can have as many of these lines as you need.

10.5 Unformatted files

The next couple of examples are fairly trivial modifications to 'non-text' files - one is an executable program file and the other is a screen dump.

Enter the command

```
        RU.flp1_edt_bin
```

Now you are using Editor to look at ... itself. Notice that you are automatically switched into Overstrike mode. Enter the command:

```
        FC.Search failed.
```

One occurrence will be found. Further tries (with F2) will yield the message "Search failed" in the error window. Now overwrite the words "Search failed" in the file as displayed on the screen, with the words "String Absent". Then enter the command:

```
        W.flp2_edt_modded. Q
```

which will write the named file and then quit from Editor. Now enter the Superbasic command:

```
        EXEC flp2_edt_modded
```

and once Editor is running, enter

```
        RU.mdv1_edt_bin. fc.Search failed.
```

One occurrence will be found. Subsequent tries (with CTRL/F2) will yield the message "String Absent" in the error window. OK, we have used Editor to create a (slightly) altered version of ITSELF, and we've then run that version.

Now for a screen dump:

Get an interesting screenful of characters with Editor, switch into Superbasic, using CTRL/C and enter:

```
SBYTES flp2_scrdump,131072,32768
```

which saves the screen. CTRL/C back into Editor and load the saved screen as an unformatted file with:

```
RU.flp2_scrdump.
```

Doesn't look at all similar, does it?. Now enter a global replacement for some character group that occurs frequently in the screen file - e.g.

```
RP EC/_*_*_*_*/zZzZzZzZ/
```

where _ represents the ASCII character 254 or 255 (you choose on inspection) and * represents the ASCII character 0. You can get the character 254 by pressing SHIFT/ENTER, the character 255 by pressing ALT/ESC and the character 0 by pressing CTRL/£.

When this is done - it will take a while, speed up the process by hitting the arrow key eight or ten times in succession - save the file with:

```
W.flp2_scrdump2.
```

Detach from Editor with CTRL/C, and restore the screen with;

```
LBYTES flp2_scrdump2, 131072:PAUSE
LBYTES flp2_scrdump, 131072
```

Interesting change?? Well, anyway you get the idea.

10.6 Procedure Directory

Assuming that playtime is over, suppose that a SuperBasic source program is loaded and it is required to produce a printed directory of each procedure and function. The printer is attached to 'ser1'.

```
bt1 t rp fc(.ine PROC.-.ine FuNc.) bs be bw.ser1
```

This sets the block type to Line and then scans from the top of file for a line marking the beginning of a function or procedure. We do not use the keyword 'DEFine' since that would find all lines of the form 'END DEFine' as well. Having found a line, it is defined as 'the block' and the block is written to the serial port - i.e. it is printed.

Suppose that you have just decided that all of your procedure and function names should be 'capitalised'. The convention might be: system keywords - UPPER case, variables - lower case and routine names - Mixed Case. Anyway, hobbyhorses apart, a similar method to the example above could be used.

```
        btl b sm rp fbc(.ine PROC.-.ine FuNc.) bs be b bl cb
```

When that finishes ("Search failed nnn"), type

```
        cm rp n 3dw
```

This will get rid of the line number and the DEFine XXX words, leaving only the routine names followed by parameters - if any. We can get rid of the parameters as follows:

```
        cm rp ec(.(.+>>)..
```

Looks a bit weird, but it says 'Find open bracket and delete from there to the end of line'. Since this is within a repeat, it will get rid of all parameter specs following the routine names.

You could then write

```
        cm rp n bs be n mm p e<<t rp ewc.< ec>>.> j
```

That says, go to the marker point, and then repeat the sequence:

> step to the next line (the routine name), take a copy of it to the next line, and turn the copy into mixed case. Return to the original line, prefix it with 't rp ewc.' and tack a dot to the end of the name, finally joining the current line and the following line.

The product of all of this is a series of lines like:

```
        t rp ewc.name1.Name1
        t rp ewc.name2.Name2   etc etc
```

Define this lot as the block, write the block to ram disc, delete the block and invoke the ram disc file with the "rc" command. It takes a whole lot longer to describe it than it does to do it!

10.7 Line standardisation

It sometimes occurs that files contain repeated patterns of lines that can usefully be joined together to standardise line formats. Examples are the output from the WSTAT command, name and address labels etc.

Suppose we have a disc (or several discs) with a lot of files, and some of the stats need examination. It would be easiest to do if each of the lines were of one type.

WSTAT gives output of the form:

```
file name
        size  date last amended
file name
        size  date last amended      etc etc
```

We would like to end up with:

```
file name     size  date last amended
file name     size  date last amended      etc etc
```

The "j" command is an obvious candidate. The only problem is that the file names are all different lengths, so we are more likely to end up with:

```
long file name    size  date last amended
file name     size  date last amended
longer file name     size  date last amended
file name     size  date last amended
```

The simple solution is to position the cursor carefully BEFORE using the 'j' command. 'j' will append the following line to the current line at the rightmost of the end of the current line or at the cursor position.

First, create the wstat file using the Superbasic commands:

```
OPEN_NEW #3,'flp1_wstatf'
WSTAT #3,'flp2_' : CLOSE #3
```

Then load the file into Editor, position the cursor to a suitable point (say column 25 to allow for long names) and join the lines:

```
r.flp1_wstatf. rp 24cr j n
```

And we end up with a sequence of neat lines, that can be sorted on name or size or date or whatever.

Using exactly the same technique, a file from a labels print, consisting of blocks of lines:

```
Name
Address 1
Address 2
Address 3
Address 4
```

could be converted into single line records, again for sorting or searching or whatever, using something like:

```
rp 29cr j 25cr j 25cr j 25 cr j n
```

A labels file, in common with many others, is likely to contain a lot of blank lines. If it is required for example to get rid of all blank lines in a file, it can be done with the sequence:

```
rp ns d p
```

This says find the next line shorter than or equal to the specified length - OK, no length was specified but the default is zero. Having found such a line, delete it. Since we are now on the next logical line of the file, and it too may be blank, recommence the scan from the prior line.

The presence of the delete command may be a bit worrying in a repeat loop, but there need be no concern, since the delete command can not be reached unless the NS command has been completed successfully. If no further blank lines exist in the file, the NS command will generate an error "Search failed".

11. Configurator

The Configurator program is provided to enable the Editor to be tailored to the user's particular requirements and machine configuration.

The program conducts a simple dialogue. To each prompt, the program provides and displays a suitable default so that the program user is aware of the desired form of the response.

The program works by reading in the whole Editor program, modifying parts of the program as specified by the user and then rewriting the Editor. As with the Editor program itself, the Superbasic extensions provided on the distribution medium must be loaded in memory.

The Configurator is started using the Superbasic command:

```
EXEC flp1_edt_config_bin
```

The program expects to find a file "edt_bin" (the Editor) somewhere. If you have renamed the Editor program file, you should change the name back to "edt_bin" before you start the Configurator.

11.1 Where is program now

"Enter the name of the device which now holds 'edt_bin': "

The Configurator needs to know where to find the Editor program that is to be reconfigured. A device name - e.g. flp1_ - should be entered.

11.2 What screen mode will be used

"Enter the run-time screen mode (4 or 8): "

There are only two responses allowed - either 4 or 8. Mode 4 is high resolution - up to 80 characters per screen line. Mode 8 is low resolution - up to 40 characters per line.

11.3 Default Data entry mode

"Enter the data entry mode (0,1 = Over,Insert): "

Each time Editor is started, it will be set in the Data entry mode that you specify here. This can be changed while Editor is running by using the F5 immediate command.

11.4 Horizontal panning

"Enter the horizontal pan percentage (10 to 90): "

When editing, if the cursor tries to go 'off the edge' of the screen, the window is panned sideways. The value you enter here indicates what proportion of 'new' text is brought into the window. The standard setting is 75% - i.e. one quarter of the columns on the screen overlap with the data displayed before the pan.

11.5 Default cursor delay

"Enter the cursor speed (1 to 3000; 1=fast): "

Once the appropriate delay figure has been established for your machine (by experiment - see 8.14.2), the figure can be 'wired in' to your copy of Editor. You may still change the delay while running Editor, should you choose to (CD command). You should select a 'faster' setting if you have a slow RAM extension board, and conversely a 'slower' setting if your extension RAM is fast.

11.6 Default block type

"Enter the type of block (1=Char,2=Column,3=Line): "

The response here determines the block type on starting Editor. As usual, it may be changed while running Editor (BT command).

11.7 Default block visibility

"Enter the visibility of block (0=Hide;1=Show): "

The response here determines the block visibility on starting Editor. As usual, it may be changed while running Editor (BH command).

11.8 What screen colours are to be used

There are 6 questions relating to the paper/ink colours of the three screen windows maintained by Editor.

After the colours have been selected, the chosen combinations are displayed, and the user may elect to respecify. The colours set here may be changed at run time (KC, KE, KM commands)

11.9 What max line length

"Enter the maximum line length: "

The run-time size of the Editor is to an extent determined by the response to this question. The default maximum size is 256 characters. The largest figure allowed is 1000.

11.10 What is the default tabbing interval

"Enter the tabbing interval: "

The response to this question may be overridden at run-time, using the 'ST' command.

11.11 What is the default right hand margin

"Enter the column of the right hand margin: "

The response to this question may be overridden at run-time, using the 'SR' command.

11.12 What character width/height

"Enter the character width in pixels: "
"Enter the character height in pixels: "

These questions are really in place to allow the Editor to run on American QLs. There is otherwise no reason to alter the values.

11.13 What maximum number of concurrent search strings

"Enter the max no of concurrent search strings: "

The number of concurrent search strings may range from 1 to 6. The default is 3. See section 8.3 for an explanation.

11.14 What is the maximum length of each search string

"Enter the max length of a search string: "

This value may range from 40 to 256. It defaults to 60. At run time, violation of this figure gives a "text too long" error message.

11.15 What name for character set file

"Enter the name of the character set file: "

The character set file contains the special font that displays the control characters. This file is optional. If the file is made available to the program, it requires about 1200 bytes of memory. If the name is changed the new name may be no longer than 12 characters.

11.16 What name for help file

"Enter the name of the help file: "

The Help file is accessed when the user presses F1 or F1/SHIFT in the Editor. If at that time the file can not be located, then no help text is displayed. If the name is changed, the new name may be no longer than 12 characters.

11.17 Where will they live

"Enter the device which will hold these files: "

The two above files are expected to live on the same device. This is usually the 'system' device - e.g. mdv1_ or flp1_ - but may be any valid device name.

11.18 Default data device

"Enter the run-time default data device: "

Again, a device identifier is required. When a file command is given in Editor's command line, if the named file can't be found, this device identifier will be used to prefix the file name. This means that you will not have to keep typing "flp2_fred"; "fred" will do just as well.

11.19 What delimiters for cursor movement/deletion

"Enter the word delimiters for cursor movement: "

When using the immediate and extended commands for 'word' movement and 'word' deletion, the definition of a 'word' is a string of one or more characters that do not include any of the characters in this delimiter string. Depending on the most frequent type of text file to be processed, there may be some advantage in adding to or reducing this list. It is of course up to the user, and the consequent action of the program in response to 'word' type commands is his responsibility. See 5.2 for more detail.

11.20 What delimiters for search by word

"Enter the word delimiters for string search: "

When using the Find and Exchange commands with the 'word' search qualifier, the definition of a 'word' is a string of one or more characters that do not include any of the characters in this delimiter string. Depending on the most frequent type of text file to be processed, there may be some advantage in adding to or reducing this list. It is of course up to the user, and the consequent action of the program in response to 'word' type commands is his responsibility. See 5.2 for more detail.

11.21 Default screen size/position

The default border, showing the overall screen area to be used by the Editor, is painted. Using the four arrow keys, the border may be moved around the screen. Using the four arrow keys in conjunction with the ALT key, the size of the window (the width and the height) may be extended or reduced. When the window is a satisfactory size and in the correct position, the ENTER key is pressed.

11.22 Updating the Editor

The configuration dialogue is now finished. The user may chose to overwrite the 'input' copy of the program, or perhaps to change the volume in the original drive (where the program was found in 11.1), and thereby create a new (configured) copy of the Editor program.

12. Quill and Quill files

12.1 Quill and Editor concepts compared

Editor and Quill have many points in common in their respective program objectives. They also have a considerable number of differences. The most significant difference between the two programs is the manner in which the objectives are achieved.

Quill is exclusively concerned with the creation, modification and printing of 'documents'. An assumption underlies the whole program, namely that Quill is the sole owner and user of the document, and that internal structural conventions within the file are entirely and exclusively the concern of Quill. "Outside world" files have to be transformed into Quill format, using the "import" facility. There is no easy or straightforward way of getting a Quill file back into outside world format (although a good effort can be made with tricky use of the Print command).

In Quill, a document is one or more sections, each with one or more pages. Each page contains one or more paragraphs, which in turn consist of one or more sentences of one or more words.

The concepts which are most strongly apparent in Quill are the document, the page, the paragraph. There is a way to 'disable' the forced paging of a document (define the 'lines per page' as zero), but the paragraph concept can not be avoided.

Editor includes within its scope document type files, but also encompasses 'non-document' files, such as program source code, data files, print files etc etc. Consequently, the top level concept of Quill - the document - equates to the Editor concept of "the file". Editor does not make the assumption that it is the 'sole user' of a file. Quite the converse, the program recognises that it has no control over the internal file structure.

As a result of the variety of file types that may be processed, the concepts of page and paragraph do not truly exist within Editor. The program tries bravely to hang on to the idea of a line, but even that concept has to be forsaken for certain types of file. The clearest and most consistent concept in Editor is the character.

Neither Editor nor Quill has a strong view of a sentence. Both however recognise the concept of a word, though the view in each program is different. Quill stays to the idea that a word is a succession of characters between two spaces (or space and start/end of line). Editor's view is similar, except that for space in the Quill definition, Editor uses an arbitrary collection of symbols and punctuation marks - arbitrary in the sense that the program user may adjust them as necessary, according to the file type, language etc that is being processed.

12.2 Philosophy of operation

The overriding concern of Quill in its interaction with the user is to make things as simple and as obvious as possible. In practice, this worthy objective is taken way over the top. Further assumptions in Quill are that the actual file size is pretty small, that the user is a steady and progressive worker and that the unexpected rarely happens. These assumptions are evidenced by the time penalty in attempting to navigate (move the cursor) from one part of a document to another - most particularly going back through a file - and the rather limited navigation commands available. All 'multi-phase' operations in Quill have their own built-in logic, from which one may never deviate. For example, 'block copy' requires that first the start of block is defined, then using a very limited range of navigation commands, the cursor must be moved to the end of the block. After that an equally limited range of navigation commands is used to find the insertion point. After the block operation is complete, Quill forgets all about the definition of the block. Similar problems exist with the "search" and "replace" commands.

During all Quill sequences, absolute priority is placed on keeping the screen looking pretty, and maintaining a verbose dialogue with the user.

These program structures are as they are presumably to avoid swamping the limited brain power of the user. Instead his patience is left in tatters, and the clock ticks on.

Editor takes an entirely different view of the interaction with the user. Certainly, the program has a preferred sequence of operation on receipt of any input (data or command) from the user. Within this sequence, modification of the 'in memory' representation of the file data is given top priority. Only after that does the program concern itself with the 'on screen' representation of the data. Editor is more than ready to be diverted from updating the screen, if new commands are being received from the keyboard. Generally, not nearly so much "hand holding" goes on during the command sequences. For example, the user may define an 'end of block' without first having defined the block start. If the user subsequently attempts to move, insert or delete the block, the program will simply complain that no proper block definition exists. Once a block has been defined, the program hangs on to the conceptual definition, so that the same (or adjusted) block may be manipulated without the task of defining it again.

Equally, the search and replace equivalents "find" and "exchange" will always start at the current cursor position, and search forward or back through the file as directed, but certainly do not reset to start of file as a matter of course.

Differences of this type are apparent through all of the comparable command sequences of the two programs. Editor assumes that the user knows basically what he wants to do and that he will do it in a manner which is most suitable to him at the time. If/when the user needs visual confirmation of his actions, he will stop typing and look at the screen as a whole, rather than merely the current cursor position. The momentary pause in typing will allow Editor to tidy up those parts of the screen that are in disarray (if any).

12.3 WYSIWYG

Quill is described as a 'What You See Is What You Get' type of word processor, meaning that the on-screen representation of the text is very close to the final form of the text when printed. Bold face, underlining, subscript, superscript, page size, top margin, bottom margin - all these are given unique display characteristics, or at least resolved into the display.

Equally, left and right margins, tab points, justification style etc are all implemented in the screen display, and "print time" features, such as header text and footer text, may be specified.

These characteristics are also 'remembered' in the document file from one editing session to another. It is for this reason that Quill requires its own 'document file format' - the formatting and highlighting information has to be stored in the file along with the text but distinguishable from the text.

From the viewpoint of user convenience, it is almost inevitable that if these formatting and highlight style requirements are to be specified by the user and remembered by the system, then the file format of the document file becomes 'non-standard'.

Because Editor is not exclusively concerned with documents, it does not directly support some of these features. This has the advantage that no 'control' (non data) information has to be included in the output file. Of all the file types that Editor can process, the only class of file which suffers any disadvantage is the pure document.

Document handling in Editor is, however, reasonably well supported, when in 'document mode'. There is dynamic display of page breaks, page and line number etc. The user may elect to include control characters to invoke highlighting features (bold, superscript etc) at print time, and these control characters will be displayed by Editor and conventionally supported (word wrap, paragraph reform, column indication, and tabbing). Other print time features - headers and footers are not directly supported, but may be achieved using conventional lines detected by the 'printer driver' program - see section 13.

12.4 Transfer of Editor files to Quill

(To the extent that any file processed by Editor can be called an 'Editor file')

A document file prepared by Editor consists of a series of (presumably) pure text characters interspersed with linefeed characters - i.e. a series of lines. This is exactly the required format of the "import" file to Quill.

If the user has entered any non-text characters into the file using Editor, the "import" command of Quill will strip them out anyway. In Quill, the import command can operate in one of two ways - by "line" or by "paragraph".

What this means is that if the file is imported by line, Quill will regard each successive line in the import file as a paragraph in its own right. If the document is a table of some kind, or contains several tabular sections, this may be what is required.

In the majority of cases however, it is likely that importing by "paragraph" is preferable. In this mode, successive lines are treated as being in the same paragraph until two linefeed characters are detected in succession. Lines considered as being in the same paragraph are therefore candidates for word wrap, justification etc etc. This is virtually exactly the definition of a paragraph in Editor, so no conflict should arise here.

It will be of considerable help in the editing in Quill if before writing the file in Editor, the paragraphs are left justified - even if the eventual required form is right justified. For similar reasons, a left margin of 1 will also help. These precautions minimise the number of spaces transferred to Quill, and this will make Quill-based editing simpler and quicker.

12.5 Transfer of Quill files to Editor

Quill files may be read by Editor in one of two ways.

The first and most obvious method is to read the Quill "_doc" file directly (using the ru command in Editor). To progress much beyond this point, you will need a fair understanding of the Quill _doc file structure, and not a little about Editor commands.

A good start is to go to the bottom of the file and scan back over all of the control information until you arrive at text that you recognise. Flip the mode to Insert (the RU will have set it to Overstrike) and press ENTER. Then define all of the back end of the file from this point as the (Line) block and delete the block.

A fair attempt at generalising the procedure thereafter is contained in the supplied command file "quill_cmd", invoked as usual with the RC command. The text box below contains notes for those interested in the principles of the command file operation.

Quill documents contain no linefeed characters. The end of paragraph marker is the ASCII code zero. The first thing to do is to 'rough cut' the text into paragraphs of reasonable proportions - approximately the right line width etc. As little work as possible should be done in 'unformatted' mode.

Do a global exchange of ASCII zero for a line split. A suitable Editor command is rp e/*//;2s where * signifies the keys CTRL and £ pressed together. Similarly, 'hard' page breaks in the Quill file are denoted by the (universal) CTRL/L. These characters should also be 'broken out' from the body of the text. With these two exchanges completed, the main paragraphs have been isolated, but are still set up as random length lines.

If the margins are set temporarily to say 1 and 79 respectively, the paragraph reform command pr may then be used to throw the text back into shape. After this, it is probably best to write the file out and read it back as a text file - the r command. On re-reading, the other format characters - CTRL/I, CTRL/L etc may be exchanged, expanded, deleted or left alone, as required. The margins may then be set to the real requirements and the paragraphs reformed.

The second and perhaps easier way of transferring a file from Quill to Editor is to "print" the document from Quill to a device (other than the printer - probably a disc or tape file).

The file created is in effect a straight text file, although some non-text characters may have been included depending on the "printer_dat" file that was available to Quill at the time that the "print" command was given. This file may be read into Editor using either the r or ru command. In the present context, there is little to chose between the two commands, unless trailing spaces on lines are important - in which case, use ru to preserve them.

Any special characters that have been included in the print operation may be stripped out or left in as the user chooses.

Equally, the (potentially) repeated headers, footers, top and bottom margins and the blank lines sometimes used by Quill to fill pages, may also be simply removed if not required.

Printing from Quill

Get Quill started and load the desired document. Then remove the Quill disk or cartridge from its drive. The reason for this is that Quill will otherwise attempt to read the file PRINTER_DAT and fill your output file with printer control characters.

Set the upper and lower margins to 0 (using Design) and set both Headers & Footers to None.

Now press F3 (command mode) and choose the option P (Print). Press Enter twice. Then enter in the name of output file. The rules for naming such files should be in the Quill documentation. The name takes the format

```
drive_identifier_extension
```

where

drive	is (say) mdv1, flp2 etc
identifier	has between 1 & 8 characters, starting with letter of the alphabet. Don't usse characters other than A-Z & 0-9.
extension	has up to 3 characters (A-Z & 0-9). It is optional. If you omit it (a) don't leave the trailing underscore after identifier; (b) Quill will add an extension _LIS to the name you choose.

Hence MDV2_SILLY_HI and FLP1_SENSIB23 are valid names, but MDV1_UNFORTUNATE is not.

You will now have an output file than can be read in using Editor. and which will require very little tidying up. We suggest you transfer ALL your Quill files in this way, in a single session. Hint: Write a command file in Editor to do the tidying up - that way you do not need to type in the commands again & again.

12.6 Comparison of EDITOR, MetaComco ED and Psion QUILL

Ratio based on EDITOR time in seconds
'-' means 'feature not supported'

	EDITOR		MetaComco ED		QUILL	
SAMPLE TIMINGS	Time	Ratio	Time	Ratio	Time	Ratio
Load file (text) *1	59.6	1	59.2	1	997	16.7
(Quill) *2	21.9	1	-		27.2	1.2
(other) *3	20.5	1	-		-	
Save file	57.6	1	151.9	2.9	91	1.6
Merge file	59.7	1	360.0	6	205.6	3.4
Create 100 lines of 64 cols	10.0	1	264.0	26	264.0	26
Define 100 line block	2.1	1	22.5	10.5	140.5	66
Move block fwd by 200 lines	5.3	1	313.1	59	227.0	42
Copy block at bottom to top	6.6	1	67.3	10.2	aaargh!!	
Delete 100 lines	3.9	1	205.6	53	120.0	31
Page from top to bottom	17.0	1	333.8	19.6	902.7	53.1
Find string - case dependant	17.3	1	17.3	1	-	
Find string	45.5	1	45.7	1	48.9	1.1
Find 10th occurrence	26.8	1	36.2	1.3	35.9	1.3
Find 12th occurrence	0.4	1	6.0	14	41.1	100
Find string A or string B	48.0	1	-		-	
Find string A and string B	45.8	1	-		-	
Find backwards	45.5	1	42.6	0.9	-	
Find string as a word	45.5	1	-		-	

*1 Test file (text) : 2600 lines 104 kbytes
*2 Test file (Quill) : 55 pages 14000 words 111 kbytes
*3 Test file (other) : 58 kbytes

FEATURES	EDITOR	MetaComco ED	QUILL
Cursor up,down,left,right	Yes	Yes	Yes
Cursor word left/right	Yes	Yes	Yes
Cursor paragraph up/down	Yes	-	Yes
Cursor to next/named page	Yes	-	Yes
Cursor to prior page	Yes	-	-
Cursor to soft page	Yes	-	-
Cursor start/end of line	Yes	Yes	-
Cursor top/bottom of screen	Yes	Yes	-
Cursor top/bottom of file	Yes	Yes	Yes
Cursor to specified line	Yes	Yes	-
Cursor to specified char	Yes	-	-
Cursor start of block	Yes	Yes	-
Cursor end of block	Yes	-	-
Cursor to marker	Yes	-	-
Cursor to last command point	Yes	-	-
Scroll screen up/down	Yes	-	-
Page screen forward/back	Yes	of a kind	-
Delete character left/right	Yes	Yes	Yes
Delete word right	Yes	Yes	Yes
Delete word left	Yes	-	Yes
Delete to end of line	Yes	Yes	Yes
Delete to start of line	Yes	-	Yes
Delete line	Yes	Yes	Yes
Delete block	Yes	Yes	Yes
Set marker	Yes	-	-
Set indent margin	Yes	-	Yes
Set right/left margin	Yes	Yes	Yes
Set tabs	Yes	Yes	Yes
Set page length	Yes	-	Yes
Show page breaks	Yes	-	Yes
Hide page breaks	Yes	-	-
Word wrap	Yes	Yes	Yes
Justify left	Yes	Yes	Yes
Justify right	Yes	-	Yes
Justify centre/middle	Yes	-	Yes
Paragraph reform	Yes	-	Auto
Overstrike/insert mode	Yes	-	Yes

FEATURES (continued)	EDITOR	MetaComco ED	QUILL
Find/Exchange forward	Yes	Yes	Yes
Find/Exchange backward	Yes	Yes	-
Find/Exchange case specific	Yes	Yes	-
Find/Exchange word	Yes	-	-
Line blocks	Yes	Yes	-
Character blocks	Yes	-	Yes
Column blocks	Yes	-	-
Move block	Yes	-	-
Retain definition of block	Yes	-	-
Sequence file on cols a to b	Yes	-	-
Renumber file	Yes	-	-
Undo current line editing	Yes	Yes	-
Undo deleted line	Yes	-	-
Issue multiple commands	Yes	Yes	-
Issue repeat commands	Yes	Yes	-
Repeat last commands	Yes	Yes	-
Recall/edit last commands	Yes	-	-
Process command file	Yes	-	-
Dynamic memory management	Yes	-	Yes
On-line help	Yes	-	Yes
Multitasking	Yes	Yes	-
Fully configurable by user	Yes	-	-
Special font for non-display	Yes	-	-
INSTANT RESPONSE TO KEYBOARD	YES	-	-

13. Printing Editor files

Files in whole or in part may be printed directly from Editor, using the W, WP and BW commands. Instead of a 'filename', the command would specify (probably) a serial port or a parallel port or perhaps a networked printer, via the network.

Examples of some suitable commands are:

```
w.ser1                        to write the whole file
gp5 bs gp7 p be bw.par        to write pages 5 and 6
wp.ser1. 5,6                  to write pages 5 and 6
bs 9n be bw.net6              to write 10 lines
```

This capability works fine as long as no special printer features are required - things such as underline, superscript etc - and there is no requirement for page headings footers and so forth.

Virtually all program listings and listing fragments, data files and many letters conform to these requirements, and so may be printed readily, directly from Editor.

This identifies two problem areas. The first is that you may not wish to wait while a file is being printed - you may have further editing to do. The second is that the document to be printed has more or less complicated print formatting or highlighting or requires headers, footers etc.

13.1 Simultaneous editing and printing

The first problem is easy to deal with. Get Editor to write the required section of the file to a temporary file area, on disc or in ramdisc, detach from Editor into Superbasic and initiate the printing. While Superbasic looks after the printing, you can reattach to Editor and continue with the editing of the same or another file.

A suitable Superbasic command might be: COPY ram1_temp_prt TO ser1
This assumes that the file or fragment written from Editor was "ram1_temp_prt" and that your printer is attached to "ser1".

13.2 Printer features

The second problem is much more difficult to solve. Part of the reason is the wide variety of printers on the market and the large range of printing facilities supported - subscript, supercript, bold, double strike, underline, emphasised, enhanced, pitches, typefaces, double width, double height, inverse, colour.

Whatever the special features are that your printer supports, there are two things that are almost certain. The first is that you will want to use many if not all of those features and the second is that the way to get your printer to do certain things will differ from many other printers on the market.

What is needed is an easy to remember convention to be used while editing a document that enables the various features of the printer to be turned on or off at the appropriate points in the document. This simple 'edit time' convention must be capable of being converted into the code sequences required by your printer at 'print time'.

The 'edit time' to 'print time' conversion table is generally known as the "printer driver". Quill uses a file called "printer_dat" to store this information. The facilities offered by Quill and printer_dat are just about adequate for a low performance, low cost printer, but fall far short of the capability of a modern multi-mode printer.

The rest of this section discusses the creation of an edit-time convention for printer control and describes a program that can achieve the translation of the edit-time convention into the required print-time character sequences.

As with most other features of Editor, complete control is in the hands of the user. The attempt has been to provide the maximum capability, so that you have the opportunity to take advantage of all of the features of your printer.

Leaving this level of control and flexibility in your hands has the consequence that you will need to know enough about the problems to understand the solutions that have been provided. So over the next few pages there is a detailed description of the evolution of the solutions available to you.

13.3 Print control characters in Editor

Since Editor will allow you to enter literally any character into a file, and to send that file to any device, it is clearly possible to encode whatever printer control codes that your printer needs directly into the document itself.

However that is probably not a good idea - remembering the right codes would be a strain and typing them would be boring.

A better solution is to invent a convention for yourself, perhaps centered around the ASCII 'control' characters. The precise details of the convention will depend on your printer, but the basic scheme may be to pick a consistent set of characters to mean certain things. For example:

Boldface	CTRL/b	Reprint with small horiz. offset
Double strike	CTRL/k	Reprint in the same position
Enlarged	CTRL/e	Print double standard width
Condensed	CTRL/s	Print half standard width
Enhanced	CTRL/h	Reprint with small vert. offset
Inverse	CTRL/m	Swap 'paper' and 'ink'
Underline	CTRL/u	
Subscript	CTRL/v	
Superscript	CTRL/a	
Italic	CTRL/i	
NLQ	CTRL/q	
Draft	CTRL/d	
Page throw	CTRL/l	
Pause	CTRL/p	
Reset printer	CTRL/r	etc etc

These characters may be typed directly into Editor, and will show up on screen in the usual way (as the capital letter with a bar over it).

Note that you should press CTRL/SHIFT/C to get CTRL/C and press CTRL/SHIFT/I to get CTRL/I.

Also, do not use the character CTRL/J. This is indistinguishable from the ENTER key and will give hopeless results. The only other key to be wary of is CTRL/Copyright - Editor regards that key as a "non break space" for paragraph formatting.

Some of the features of a printer require an 'on' trigger and an 'off' trigger - underline for example. Some features are logically exclusive, and so form 'off' triggers for each other - the 'off' trigger for NLQ may for example be 'draft' and vice versa. Some features do not require an 'off' trigger - page throw for example.

Using only the codes suggested so far, there is - at edit time - an ambiguity between an 'on' code and an 'off' code for a feature. For example one occurrence of CTRL/B means 'bold on' and the next means 'bold off'. You may decide that you need different conventional codes for 'off' triggers, so that you get a positive indication of each action. In this scheme, CTRL/B could mean 'bold on' and CTRL/N CTRL/B could mean 'not bold' or 'bold off'.

So a revised scheme for the edit-time codes could become:

	ON	OFF	
Boldface	CTRL/b	CTRL/n	CTRL/b
Double strike	CTRL/k	CTRL/n	CTRL/k
Enlarged	CTRL/e	CTRL/n	CTRL/e
Condensed	CTRL/s	CTRL/n	CTRL/s
Enhanced	CTRL/h	CTRL/n	CTRL/h
Underline	CTRL/u	CTRL/n	CTRL/u
Subscript	CTRL/v	CTRL/n	CTRL/v
Superscript	CTRL/a	CTRL/n	CTRL/a
Inverse	CTRL/m	CTRL/n	CTRL/m
Italic	CTRL/i		
NLQ	CTRL/q		
Draft	CTRL/d		
Page throw	CTRL/l		
Pause	CTRL/p		
Reset printer	CTRL/r		

This means slightly more typing at edit-time, and a slightly more 'ugly' appearance to the text on screen. On the other hand, there is now positive indication of the 'on' and 'off' actions. The earlier convention where the same code is used for the 'on' and 'off' effects (a "flip-flop") also requires that the mechanism that converts edit-time codes to printer control sequences should be a bit "smarter".

Yet another factor that varies from one printer to another is which features (if any) are switched off at end of line. This must not be allowed to become a consideration at edit-time. If for example your printer automatically turns off boldface at the end of a line, and in your file a line contains a section of boldface text - with an 'on' code before and an 'off' code after - the editing user should not have to worry if this section gets split over two lines through subsequent editing, caused by word wrap or paragraph reform etc. Again, the problem should be solved by the conversion mechanism, which appears to be getting smarter all the time.

13.4 Conversion of edit codes

The edit code convention that has been invented will make more or less demands on the 'intelligence' of the conversion process.

What is needed in principle is a series of 'exchanges' to swap the edit-time codes for the appropriate printer control codes. This kind of thing can be handled by an Editor command file - but only if the interaction between printer facilities is minimal. Again, special action would be needed for page headers and footers and page numbering and so forth.

For anything other than the most basic requirement, the command file approach is probably not the best way to go.

Conversion by command file

As a one-time operation, build a 'command file' that contains a series of global replace statements - replace all CTRL/b with whatever your printer needs, similarly CTRL/u etc etc. Depending on the codes that you are changing to, you may need to be careful about the sequence of the commands. Optionally, the last command in the command file could be to write the edit file to the serial or parallel port.

When you have finished editing your document or text file, save the updated file and then invoke the command file (with the rc command). All of your conventional printer control commands will be converted into the real ones, and then the file may be printed directly.

Headers and footers, top and bottom margins could also be achieved in a similar fashion, as could forced page breaks etc etc. The following command will put a page break and heading every 55 lines of a file:

```
        t; sl 1; sl 1; rp 1/*Heading Line/; 56n
```

- * is CTRL and L pressed at the same time.

It may also be convenient to separate the conversion and printing phase from the Editor, so that editing may continue while some printing is taking place.

Several methods are available, but probably the best route is to create a separate program, that integrates the conversion and printing processes. This program needs to be aware of many aspects of how your printer works, and what convention you have adopted for edit-time control. To establish the detail design of the program, you will need to have available:

Your sketched out edit-time conventions
Your printer's manual (control codes section)
QL User Guide; Concepts section; Character set

13.5 A simple printer driver

For the purpose of this example, let us assume that you want to use two modes in your printouts - Emphasised print, and Condensed print. Let us say that when you examine your printer's instruction manual you find:

Emphasised:	ESC E	(Character code 27 followed by capital E)
Condensed:	SI	(Character code 15)

```
1000 REPeat main_lp
1010   INPUT "Enter name of file to be printed: ";nam$
1020   OPEN_IN #4, nam$
1030   OPEN #5, "SER1"
1040   REPeat file_lp
1050     IF EOF(#4) : EXIT file_lp
1060     INPUT #4, a$
1070     Check_for_Pause CHR$(16)
1080     PRINT #5, Translated$(a$)
1090     END REPeat file_lp
1100   CLOSE #4: CLOSE #5: BEEP 9999,9999
1110   END REPeat main_lp
1130 DEFine FuNction Translated$(txt$)
1140 LOCal wk$
1150   IF txt$ = "" : RETurn ""
1160   wk$ = txt$
1170   IF Changed(CHR$(2), CHR$(27)&"E") : RETurn wk$
1175                         REMark  CTRL/B to ESC E
1180   IF Changed(CHR$(19), CHR$(15)) : RETurn wk$
1185                         REMark  CTRL/S to SI
1190   REMark any other exchanges go in here
1400   RETurn txt$
1410   END DEFine
1430 DEFine FuNction Changed(srch$, repl$)
1440 LOCal pre$, post$, x%
1450   x% = srch$ INSTR wk$
1460   IF x% = 0 : RETurn 0
1470   IF x% = 1
1480     pre$ = "" : ELSE
1490     pre$ = Translated$(wk$(1 TO x%-1)) : END IF
1500   IF x% = LEN(wk$)
1510     post$ = "" : ELSE
1520     post$ = Translated$(wk$(x%+1 TO LEN(wk$))) : END IF
1530   wk$ = pre$ & repl$ & post$
1540   RETurn 1
1550   END DEFine
1570 DEFine Check_for_Pause(srch$)
1580   REPeat ps_lp
1590     x% = srch$ INSTR a$
1600     IF x% = 0 : EXIT ps_lp
1610     IF x% <> 1 : PRINT £5, Translated$(a$(1 TO x%-1));
1620     BEEP 9999,9999
1630     INPUT 'Print paused - press ENTER to continue ';p$
1640     IF x% = LEN(a$) : a$ = "" : EXIT ps_lp
1650     a$ = a$(x%+1 TO LEN(a$))
1660     END REP ps_lp
1670   END DEFine
```

The outline listing above will accomplish the changes from the the suggested edit-time convention for Emphasised and Condensed to the required printer control codes (and as many others as you care to add). It also shows how to incorporate a "pause" in the printing - you may want to pause at page throw or for daisy wheel change etc. It does not do anything about the 'off' codes, nor solve any problems regarding automatic feature 'off' at end of line, nor headers, footers, page numbers and so on.

13.6 The supplied printer driver

The program sketched out above is the 'baby brother' of a much more competent printer driver supplied with Editor. The program is called "edtprt_bin", and like Editor is compiled.

It obtains all of its information about your edit-time conventions and about your printer from a file called "driver_dat". This file is analogous to Quill's "printer_dat", but differs in that it is open-ended - you may put into the file as many or as few exchanges as you require. The "driver_dat" file may be created using Editor.

Edtprt_bin is a full-function printer driver which does the edit-time convention substitutions, takes account of end of line problems, and which additionally allows various types of page numbering, multiple multi-line headers, footers, partial prints, reprints etc.

The key feature of the program is that it is 'driven' by a reasonably free format "printer description" file. Within this file, you may easily identify all of the features of your printer that you wish to use. The description file may contain many sets of printer descriptions - as many as you wish - so that control information for all of your printers (or any that you are likely to use) may be contained in the one file, selectable at run time with one key press.

Much of the information for a printer is "static" - that is, it need only be said once. For example, how to set the printer in bold mode, or how to set the left/right margin etc. This is the kind of thing that is specified in the description file.

Some of the information for a document is "static". For example, what is the intended page length, what header margin should be used, what page headings/footings should be used. However, some of the document information is not static. For example, when should underline be switched on, when to change into/from italics and so forth. EDTPRT allows all of this kind of information to be embedded within the document file itself. It uses a system of conventions which are completely compatible with the "document mode" of Editor.

13.7 Running the program

To run the program, you will need to have two things. One is a file to be printed. The other is a description of the printer you intend to use.

Before you run the program, you should have set up a description of your printer. To do this you will need to look a the section below dealing with the parameters and parameter formats.

The program starts up by painting its standard screen, on which are displayed the 5 principal options:

F1	Specify Default Device
F2	Specify Printer Device
F3	Select Printer Type
F4	Commence Printing
F5	Quit

The current settings for the first 2 items are shown. For example, the default device may be shown as "FLP1_" and the printer device shown as "ser1".

The first thing that you are likely to want to do is to select the printer type. Now the printer type (and its description) is obtained by the program from your printer description file. This file is expected to be called "driver_dat", but you may call it what you like. When you press F2, the program asks you to confirm that the name of the file is "driver_dat" and that it is on the specified default device. If the name is different, then simply type in the name you have used.

The description file is read and all of the printer types it contains are displayed on screen; you select the one you want.

If you wish to print to "ser2" or "par_16k" then press F2 and enter the required value to the displayed prompt.

Changes to the default device, printer type, printer device will cause the displayed settings within the menu to be altered to reflect the current values.

When these things have been set up as you want them, press F4 to start printing the file that you want. The first thing that happens is that the parameters from the printer description file are read into memory to initialise all of the controls for your printer. The program will then ask for the name of the file to be printed (it is expected to be on the default device).

Having located the file, the next prompt asks for the start page number (it defaults to 1 of course). This is followed by the finish page number (defaulting to 'end of file'). When you have replied to that, you are under way.

During printing a message is displayed on-screen informing you that if you want to terminate the printing NOW you should press the ESC key. If you want the printing paused prior to the next "page throw" you should press the letter "P". This latter feature allows you to change continuous stationery, realign etc. [Cut sheet users have another way of halting the printing after each page - see the parameters].

And that basically is it. You can print any number of files in succession, simply by pressing F4 - assuming that none of the other parameters are required to change in the meantime.

13.8 Printer description

The printer description file contains control information describing a number of printers - well, at least one. Each description commences with a "Name" parameter, and all statements after the Name parameter and before the next Name parameter (or end of file) are considered part of the description.

All statements are contained on one line.

Each statement line starts with semi-colon in column 1. Any line which does not start with semi-colon is assumed to be a comment line.

After the semi-colon must follow a parameter type. There may be zero through many spaces between the semi-colon and the parameter type. The parameter type should be one from the list below - if it is not, then the line will be ignored without comment. All parameters other than the "Name" parameter start with the letter "P". (A common description file may be used to specify the GRAPHICS capabilities of the printer as well as the TEXT capabilities. In the present instance, we are interested only in the TEXT capabilities, whereas the program "graFix" is interested in the GRAPHICS capabilities).

Each of the parameters types addresses a different aspect of printer control or report formatting. The following list contains ALL of the parameter types supported. In looking through this list it may help (or confuse) to remember that these parameters MAY occur in the printer description file OR they may occur as embedded commands in the document file to be printed - or they may occur in BOTH. Some of the commands make more sense coming from the description file, while others are more appropriate in the document file itself. This distinction will be made clearer a bit later.

For the time being here are the parameter types:

Name	Introduces a new printer description set	Up to 30 characters - free format
PDV	Device. To which device on the computer is the printer connected.	Default: SER1
PBD	Baud rate. The speed at which the printer is set to receive characters from the computer	Default: Current setting
PRS	Reset sequence. A control sequence which causes the printer to reset to its internal defaults.	Default: None
PCN	Cancel sequence. Causes all data in the printer buffer to be lost	Default: None
PPA	Preamble sequence to be sent to the printer at the beginning of the document (fragment).	Default: None
PPO	Postamble sequence to be sent to the printer at the end of the document (fragment).	Default: None
PHM	Header margin (number of lines) to be reserved at the top of each page. Any "page header" specified will occur within these lines.	Default: 3
PFM	Footer margin (number of lines) to be reserved at the bottom of each page. Any "page footer" specified will occur within these lines.	Default: 3
PPL	Useable page length (number of lines). The amount of space within which document text will be printed.	Default: 60
PPG	Page numbering required. Implies no page number is requested in the header lines or the footer lines (if any such lines provided), BUT a page number is required in the DEFAULT position	Default: None
PPC	Column number for default page number	Default: 40; only used if default page numbering in operation

PPN	Reset page number (new page number). May occur anywhere within a document. Typically used at the start of a doc. "First doc. page is page XXX"	Default: 1
PLM	Left margin (column number). Specifies where on the physical page printing will start. Users of 80 column printers should be cautious when using this feature	Default: 1
PEL	End of Line code. Specifies the control sequence to be sent at each end of line.	Default: #13, #10
PSS	Single Sheet printing required. The program halts after a page pending a key press.	Default: No
PPS	Pause code specifier. If a pause code is met in the document file a message is displayed and the program halts pending a keypress	Default: None
PPB	Conditional Page Break (number of lines). If the number of useable lines remaining on the current page is less than the specified number, a page break will occur.	Only meaningful within the document file
PHD	Page header text - see section on headers and footers.	Default: None
PFT	Page footer text - see section on headers and footers.	Default: None
PBO	Boldface is not supported by the printer. The program should 'synthesize' boldface.	Default: No
PUL	Simultaneous underline is not supported by the printer. The program should 'synthesize' simultaneous underline.	Default: No
PXG	Exchange with Global effect See Exchanges and Toggles	
PXL	Exchange with Line effect See Exchanges and Toggles	
PTG	Toggle with Global effect See Exchanges and Toggles	
PTL	Toggle with Line effect See Exchanges and Toggles	

13.9 Parameter formats

In the brief syntax descriptions below, the following meanings apply:

String	a collection of characters commencing in a non-space, non-comma character and containing any characters
Numeric	A series of characters from the group 0 to 9
Compound string	A string built up from one or more elements. Each element may be either a SINGLE ASCII character specification of the form #nnn OR the element may be a string. Elements are separated by a comma. A more complete description occurs in 'Exchanges and Toggles'.

If there are more than one arguments on a parameter line, then the arguments are separated by semi-colon.

```
;Name String - Name of printer
;PDV  String - Device code
;PBD  Numeric - baud rate
;PRS  Compound string - max length 30 characters
;PCN  Compound string - max length 30 characters
;PPA  Compound string _ max length 30 characters
;PPO  Compound string _ max length 30 characters
;PHM  Numeric - number of lines of top margin
;PFM  Numeric - number of lines of bottom margin
;PPL  Numeric - number of useable lines on page
;PPG  None
;PPC  Numeric - column number where "Page: nnn" is printed
;PPN  Numeric - new page number
;PLM  Numeric - column number to be used as col 1 of printout
;PEL  Compound string - end of line code
;PSS  None
;PPS  Compound string
;PPB  None
;PHDn String - Header text for header line 'n'
;PFTn String - Footer text for footer line 'n'
;PBO  Compound string; Compound string
      - Boldface ON_mnemonic; Boldface OFF_mnemonic
;PUL  Compound string; Compound string
      - Underline ON_mnemonic; Underline OFF_mnemonic
;PXG  ident; ON_mnemonic; ON_print; OFF_mnemonic; OFF_print
      See Exchanges and Toggles
;PXL  ident; ON_mnemonic; ON_print; OFF_mnemonic; OFF_print
      See Exchanges and Toggles
;PTG  ident; ON/OFF_mnemonic; ON_print; OFF_print
      See Exchanges and Toggles
;PTL  ident; ON/OFF_mnemonic; ON_print; OFF_print
      See Exchanges and Toggles
```

13.10 Headers and Footers

The 'header' for a document MAY be specified within the description file, but it is more likely to occur within the document file itself. Multiple lines of header and/or footer may be specified - up to 9 lines of text for either. The parameter type PHD or PFT is IMMEDIATELY followed by a numeric character 1 to 9. The number indicates which line of text is being specified. If the parameter type PHD0 or PFT0 is specified, then the current header or footer is discarded - no header/footer will be printed. This feature is provided as a convenience.

After the parameter type PHD1 to PHD9 or PFT1 to PFT9 there follows the text of the header line. If the first character of the text is space, the space is deemed NOT to be part of the text - serving instead merely as a separator between the parameter type and the line text. So, the two following lines have the same effect:

```
;PHD3 Subordinate Information
;PHD3Subordinate Information
```

In each case the "S" of Subordinate will be the first character of the line printed. Whereas

```
;PHD3  Subordinate Information
```

will result in the first character of the line being space and the SECOND character is the "S".

The first line of the header and/or footer may optionally contain a page number. The requirement for a page number is indicated by the presence of the character "#" on the line. There are three principal forms for the page number:

- A single # followed by any character (or end of line) other than another # or "R", indicates that the page number is to be printed LEFT JUSTIFIED at the character position indicated by the # mark.

- A series of # characters in succession indicates that the page number is to be printed RIGHT JUSTIFIED in a space denoted by the series of # marks.

- A single # followed by the character "R" or "r" indicates that the page number is to be printed LEFT JUSTIFIED at the character position indicated by the # mark, the display to use ROMAN numerals.

For example the three specifications below:

```
     ; PHD1Standard heading text      Document ref.     Page: #
     ;PHD1 Standard heading text      Document ref.     Page: ####
     ; PHD1 Standard heading text      Document ref.     Page: #R
```

will - for page 14 - produce the following output

```
     Standard heading text      Document ref.     Page: 14
     Standard heading text      Document ref.     Page:   14
     Standard heading text      Document ref.     Page: xiv
```

13.11 Heading and footing position

Headings are printed within the header margin. Footers are printed within the footer margin. If the number of lines of header/footer text exceeds the header/footer margin, the margin size is increased to at least accomodate the text. If the margin size exceeds the number of text lines, a blank line will occur at top of page before the header is printed, and/or a blank line will occur after the footer is printed. This is to avoid the perforations. Any lines in excess of header+1 occur between the last header line and the main text of the page. Similarly for footers.

13.12 Print starting from a nominated page

Printing from a particular page number needs one or two comments.

First, since you are allowed to change the page number within a document (using the PPN parameter), the start page number quoted MAY not be unique. The program will use the LOGICAL page number - i.e. the one that is printed on the page - to identify the page you want. You will be aware if you use the PPN command, and will have to watch out for whatever ambiguity that might entail.

Second, it is fairly common for 'font settings' to be made right at the start of a document, and never mentioned again thereafter. In fact the same could be true of any print formatting feature. Further, if you are to start printing some fair way through the document, other features may have been changed en route - e.g. header or footer text, font, character spacing, line spacing and so on. A further point is that conditional page breaks may have been specified within the document. All this leads to the situation that the whole document has to be scanned in some detail to establish (a) where actually is the required page start and (b) what printer features, modes etc should actually be in effect when printing commences.

The result is that "start from page 31" may go quiet for a while before printing starts - the program is reviewing the previous 30 pages.

Note also that even though a page number may be output in Roman numerals, it is the decimal version of the page number that is required as a "start from page" specifier.

13.13 Exchanges and Toggles

Your printer may support many text enhancement features. The more common among these are Boldface, Underline, Subscript, Superscript. Some others are Italics, Draft font, NLQ font(s), Compressed, Enlarged, Inverse, Colours, 10 characters per inch, 12 cpi, 6 lines per inch, 8 lpi etc etc.

Which features are available varies from one printer to another. (Whether you want to use the features is also up to you to decide). When different printers offer similar features, it is often NOT the case that the two printers require the same sequence of control characters to achieve the same effect.

In any case, the control sequence required by the printer to switch into, say, Subscript mode might be quite complicated and consist of very obscure characters.

So, it is convenient in the document to ignore what the printer actually needs and to use a system of mnemonic codes to switch on and switch off the various printer features as they are required. The exchange options provided within this program serve the purpose of converting the "edit-time" mnemonics into the "print-time" control sequences.

Some of the features of a printer MAY automatically be switched off at the end of a line (the printer manual will clearly state this). A typical feature of this kind is Enlarged (Double Width). Many or most of the features will stay ON from when they are set on until they are explicitly set off.

For various reasons, it is necessary for you to indicate in the printer description which features are of the former kind - referred to as Line effect features - and which features are of the latter kind - referred to as Global effect features.

Within your document, you will probably want to use a system that requires you to turn on a feature EXPLICITLY and to turn it off EXPLICITLY. It would be dangerous to leave it to the printer to turn a selected feature off - it requires that you remember which features DO get turned off, and in any case this might differ from one printer to another, so swapping documents electronically could become a problem if the recipient has a different type of printer.

Assuming that that is agreed, then for each of the features of the printer it is necessary to invent an ON code (for edit-time use) and an OFF code. You might decide that to turn on Boldface, you will key the character CTRL/B. In Editor, this character will be displayed as a capital B with a bar above it. Editor will also know that it is a print control mnemonic and does not count toward the line length, nor does it advance the column position.

Now you may decide that you would like a different code to switch off Boldface. You might chose CTRL/O (for Off). There are two immediate consequences. The first is that other features are going to need off codes, and you can't (won't want to) use CTRL/O for all of them. The second is that there is little intuitive connection between CTRL/O and Boldface - it is something that you have to remember.

So instead, you may decide that you would like CTRL/B to switch on Boldface AND to switch off Boldface. CTRL/B then becomes a "toggle". Toggles are great to use - the system is neat, and there is a minimum number of codes to remember - but they have one significant disadvantage. If you forget just once to switch off something (Boldface, Underline or whatever), the rest of the document is printed as a mirror of that feature - the next toggle is supposed to switch the feature ON but instead switches it off and so forth. That is where explicit on codes and off codes score.

Anyway the choice is yours, since the print program supports both methods. As a summary of the two methods, here are some examples of how the Boldface feature may be defined:

Description file fragment (explicit ON/OFF):

```
    ;PXG Boldface; #2;  #27, E;  #14, #2;  #27, F
    ; Use CTRL/B to switch ON - generates ESC E
    . Use CTRL/N CTRL/B to switch off - generates ESC F
```

Document fragment:

```
    the next bit {CTRL/B}is in Boldface{CTRL/N}{CTRL/B} now it's off
```

Description file fragment (toggle ON/OFF):

```
    ;PTG Boldface; #2;  #27, E;  #27, F
    . Use CTRL/B to switch ON/OFF - generates ESC E, ESC F alternately
```

Document fragment:

```
    the next bit {CTRL/B}is in Boldface{CTRL/B} now it's off
```

The program supports two types of Exchange and two types of Toggle. In each pair the difference is that one is for Global effect features and the other is for Line effect features.

Exchange format:

```
    ;para_type ident; ON_mnemonic; ON_print; OFF_mnemonic; OFF_print
```

para_type	is either PXL (line effect) or PXG (global effect).
ident	is some string that reminds you what the parameter does
ON_mnemonic	is whatever compound string you have in the document that needs to be replaced.
ON_print	is the printer control sequence that it should be converted into.

OFF_mnemonic and OFF_print are optional. They would be omitted if a conversion sequence was needed to print, for example the pound sign £ or the hash mark # or umlaut or any other special character.

Toggle format:

```
    ;para_type ident; ON/OFF_mnemonic; ON_print; OFF_print
```

para_type is either PTL (line effect) or PTG (global effect). The others are as above. None is optional.

13.14 Specifying compound strings

Some of the information required for the parameters contains some strange characters, as you will see from your printer manual.

Rather than have you searching around for some weird key combinations on the QL, we have set up the string conventions so that information in your printer manual may be entered directly into the driver file.

Characters in parameter strings may be expressed either as a letter or sequence of letters - which will be used exactly as they appear (e.g. UPPER CASE or lower case) - or as a decimal number representing the ASCII code for the (single) character to be used.

Combinations of letters and decimal numbers are allowed. A comma must be used as a separator. Leading spaces in front of each element of the string are ignored. A decimal number must be prefixed by #. If the # character is required as a letter, it must occur as ##. Neither a comma nor a semicolon character may used in a string of letters. If either is needed, instead use #44 or #59 respectively. Equally, a string may not BEGIN with a space - use #32 if the requirement arises.

So, for example, to set up a string as

```
    ESC "E" ESC "T14" ESC ";" ESC " AB,"
```

the compound string in the parameter line would look like:

```
    #27, E, #27, T14, #27, #59, #27, #32, AB, #44
```

13.15 Built-in exchange

There is one substitution built-in to the program. That is the "non-break space" feature of Editor. If the code ASCII 31 occurs in the document to be printed it will be converted into a space.

The effect is as though the following parameter line occurred in every printer description set:

```
    ;PXL Non break space; #31; #32
```

13.16 An example parameter set

```
;Name Toshiba TH2100H Font 1
;PPA #27, U06, #27, F1
. set to 6 lpi; default font 1
;PPO #27, U06, #27, F0
. reset to Draft font (font 0)
;PRS #00
;PDV ser2
;PEL #13,#10
;PBO #15; #15
.  Boldface uses program default - no feature on printer
.  CTRL/O is used for both ON code and OFF code
;PTG Boldface; #15;
. "on" code and "off" code are null - 'cos printer does not
. support simultaneous Boldface.  So "PBO" parameter is used
. to tell program to simulate Boldface - using backspace & reprint
;PTG Underline; #21; #27, X; #27, Y
. simultaneous u/line supported, so use CTRL/U as toggle code
;PXL Hash sign; #35; #163
;PXL Pound sign; #96; #35
. above specified as 'straight' translates
. - Line effect and no "off" code meaningful
;PXG Draft font; #4; #27, F0
;PXG Courier NLQ; #6,#1; #27, F1
;PXG Gothic NLQ; #6,#2; #27, F2
. above specified as 'mode setters'
. - no "off" codes, but with Global effect
;PTG Superscript; #22; #27,U12,#27,M; #10,#27,U06
. superscript not supported, so set to 12lpi and do
. single reverse linefeed before superscripted printing
. After printing, do linefeed and cut back to 6lpi
;PTG Subscript; #20; #27,U12,#10; #27,M,#27,U06
. similar method to superscript
```

15. Command Summary and Index

Immediate commands:

Cursor Controls:

	Move		Delete	
Character :	LEFT	RIGHT	CTRL/LEFT	CTRL/RIGHT
Word :	SHIFT/LEFT	SHIFT/RIGHT	CTRL/SHIFT/LEFT	CTRL/SHIFT/RIG
End line :	ALT/LEFT	ALT/RIGHT	CTRL/ALT/LEFT	CTRL/ALT/RIGHT
Line :	UP	DOWN	CTRL/SHIFT/ALT/LEFT	
Line :	CTRL/DOWN	ENTER	CTRL/LEFT (col 1)	
Screen :	SHIFT/UP	SHIFT/DOWN		
Scroll :	ALT/UP	ALT/DOWN		
Screen page:	SHIFT/ALT/UP	SHIFT/ALT/DOWN		
Start parag:	CTRL/ALT/UP	CTRL/ALT/DOWN		

Temp Margin	
Left margin:	ALT/TAB

Other immediate commands:

ESC	Interrupt commands (if commands executing)
F2	Re-execute prior command
F2/CTRL	Re-execute prior Find/Exchange command
F2/SHIFT	Edit and execute prior Find/Exchange command
F3	Specify Command group and execute
F3/SHIFT	Edit and execute prior command
F4	Redraw Screen
F4/SHIFT	Resize screen
F5	Toggle data entry mode between Overstrike and Insert
F5/SHIFT	Collect garbage

14. System Limits

File size:

Characters:	Limited by the size of RAM
Words:	No limit
Lines:	32767 maximum
Paragraphs:	No limit
Pages:	No limit (each page counts as 1 line in line limit)

Screen size:

Lines:	Maximum: 24 data + status
	Minimum: 4 data + status
	Default: 19 data + status
Columns:	Maximum: 84 (+border)
	Minimum: 20 (+border)
	Default: 60 (+border)
Panning:	Maximum: 100%
	Minimum: 10%
	Default: 75%

File text:

Line length:	Maximum: 1000 characters
	Minimum: 60 characters
	Default: 256 characters

Command line strings:

Command line:	Maximum: Line length (as above)
Strings:	Maximum: Line length
	Default: 60 characters

Extended commands:

The symbols in the summary below have the following meanings

*	= Number must follow
**	= String must follow
*?	= String is optional
,	= One or two optional numbers may follow

A	Insert line after current line **	
AF	Insert file from device after current line **	
AP	Ask for Parameter value (within command file only)	
B	Cursor to Bottom of file (synonym GB)	
BD	Delete block	
BE	Mark current line as end of block	
BH	Block Hide (or block show if not showing)	
BI	Insert block after current line	
BM	Move Block to new position after the current line	
BS	Mark current line as start of block	
BT	Specify Block Type (as Char, Kolumn, Line)	
BW	Write block to device **	
BZ	Sound the Buzzer	
CD	Specify Cursor Delay time *	
CB	Cursor to start of Block	
CE	Cursor to End of line	
CK	Cursor to end of blocK	
CL	Cursor to one char Left	
CM	Cursor to Marker point	
CP	Cursor to next Paragraph	
CR	Cursor to one char Right	
CS	Cursor to Start of line	
CW	Cursor to Word right	
D	Delete current line	
DC	Delete current character	
DW	Delete current word	
E	Exchange strings *?	Qualifiers: BCWQ
EX	Execute the current line	
F	Find string *?	Qualifiers: BCW
GB	Cursor to Bottom of file	
GC	Cursor to named Character *	
GL	Cursor to named Line *	
GP	Cursor to next or named Page *	(Document only)
GPB	Cursor to prior Page *	(Document only)
GPS	Cursor to Next 'soft' Page *	(Document only)
GT	Cursor to Top of file	
I	Insert line before current line **	

J	Join line with next	
JC	Justify Centre	
JL	Set Justify left mode	
JM	Justify Middle	
JR	Set Justify right mode	
KC	Specify Ink/Paper for Command screen *,*	
KE	Specify Ink/Paper for Error screen *,*	
KM	Specify Ink/Paper for Main screen *,*	
L	Cursor to Last command point	
LL	Specify (average) Line Length *	
M	Specify Memory size *	
MD	Make document from text file	
ML	Cursor word to lower case	
MM	Cursor word to mixed case	
MU	Cursor word to upper case	
MR	Extend the right margin	
N	Cursor to Next line	
NL	Cursor to Next Longer line *	
NS	Cursor to Next Shorter line *	
P	Cursor to Prior line	
PH	Page Hide/Show	(Document only)
PL	Set (document) Page Length	(Document only)
PR	Paragraph reformat	
Q	Terminate w/out save	
R	Read text file from device **	
RC	Read command file from device **	
RD	Read document file from device **	
RN	Renumber all lines *,*	
RP	Endless repeat of commands	
RU	Read unformatted file from device **	
S	Split current line at the cursor point	
SH	Show system status	
SI	Set indent margin posn *	
SL	Set left margin posn *	
SM	Set Marker to current line	
SQ	Sequence the lines in the block *,*	Qualifiers: N
SR	Set right margin posn *	
ST	Set TAB increment value *	

T	Cursor to Top of File (Synonym GT)
TA	Specify Tab stops Assymetric *,*,.....
TC	Compress current line using Tabs
TD	Delete Tab stop *
TE	Expand current line Tab characters
TI	Insert Tab stop *
TR	Remove all Tab stops
U	Undo changes to this line
UD	Restore last Deleted line
W	Write whole file to device *?
WP	Write whole or part of document to printer
WR	Write whole file to device, making filename the default **
X	Terminate with save
Z	Zap the current file in memory