FULL SCREEN EDITOR

ED Manual

		Page
1.	**INTRODUCTION**	1
2.	**TUTORIAL GUIDE**	3
2.1	Getting Started	3
2.1.1	The Status Line	5
2.1.2	The Command Line	7
2.1.3	Text Area	8
2.2	Appending New Text (F1 key)	8
2.3	Appending or Inserting Lines (F1/F2 keys)	9
2.4	Using the Del and Back-arrow keys	9
2.5	Inserting text using the Ins key	10
2.6	Other Cursor keys which Simplify Editing	10
2.7	Saving your Text	11
2.8	Exercise 2	11
2.9	More on the F1 and F2 keys	12
2.10	Deleting Lines (F3 key)	13
2.11	Filling Lines (F4 key)	14
2.12	Centering Lines (Ctrl-F4 key)	14
2.13	Splitting and Joining Lines (F5/F6 keys)	14
2.14	Tagging Blocks of Text (F7/F8 keys)	15
2.14.1	Line Tagging	15
2.14.2	Block Tagging	16
2.14.3	Insert Mode and Block Move and Copy	18
2.14.4	Re-setting the Last Tagged Lines or Block	18
2.15	Re-executing Commands (F9/F10 keys)	18
2.16	Zooming Your Text	19
2.17	Tabs	19
2.18	Composed Characters	20
2.19	Line Drawing Characters	20
2.20	Margins	20
2.20.1	Moving Your Margins (Shift F1 to F6)	20
2.20.2	Auto Fill and Your Right Margin	20
2.20.3	Auto Justify	21
2.20.4	Indenting and Your Left Margin	21
2.21	Line Flags	22
2.21.1	Overstrike Flag (Alt-o)	22
2.21.2	Continuation Flag (Alt-c)	22
2.21.3	Paragraph Flag (Alt-p)	22
2.22	Some Simple Editor Commands.	23
2.23	Learn Mode	23

2.24	Absolute Line Positioning	23
2.25	Simple Pattern Matching	23
2.26	File I/O Commands	25
2.27	The View Command	28
2.28	Executing System Commands	28
2.29	Epilogue	29
3.	**Using Ed on a Terminal**	31
3.1	Setting Your Terminal Type	31
3.2	Required Terminal Capabilities	32
3.3	Screen Output	32
3.4	Keyboard Input	33
3.5	QNX Compatible Terminals	35
4.	**REFERENCE MANUAL**	37
4.1	The Syntax of Editor Commands	37
4.1.1	Line Range	37
4.1.2	Command Specification Character	39
4.1.3	Right Arguments	39
4.2	Placing Multiple Commands On A Line	40
4.3	Special Characters	40
4.3.1	The Newline Character (hexadecimal 1E)	41
4.3.2	The Null Character (hexadecimal 00)	41
4.3.3	The Meta Characters @$&.*[	41
4.3.4	The Backslash Character \	41
4.3.5	The Tab Character (hexadecimal 09)	41
4.3.6	The Command Character (hexadecimal FF)	42
4.3.7	The Recall Character (hexadecimal FE)	42
4.3.8	The Keyboard Input Character (hexadecimal FD)	42
4.3.9	The Macro Disable Character (hexadecimal A3)	42
4.4	The Condition Register	43
4.5	Delete Buffers	43
4.5.1	The Character Delete Buffer	43
4.5.2	The Line Delete Buffer	43
4.6	Break Handling	44
4.7	The Pattern Matcher	45
4.8	Some Pattern Examples	47
4.9	Editor Commands	48
4.10	a - Append After Current Line	49
4.11	b - Branch	50
4.12	c - Change Lines	51
4.13	d - Delete Lines	52

4.14	e - Edit a New File	53
4.15	f - File Query/Set	54
4.16	g - Global	55
4.17	i - Insert Before Current Line	57
4.18	j - Join Two Lines	58
4.19	k - Kopy Lines	59
4.20	l - Learn	60
4.21	m - Move Lines	61
4.22	o - Option Query/Set	62
4.22.1	oa - Option Anchor	62
4.22.2	ob - Option Blank	62
4.22.3	oc - Option Command	63
4.22.4	od - Option Dual	63
4.22.5	oe - Option Environment	63
4.22.6	of - Option Fill	63
4.22.7	oi - Option Insert	63
4.22.8	oj - Option Justify	63
4.22.9	ol - Option Limit	64
4.22.10	om - Option Meta Characters	64
4.22.11	on - Option Newline	64
4.22.12	os - Option Autosave	64
4.22.13	ot - Option Tabs	64
4.22.14	ow - Option Wrap	64
4.23	p - Print Lines	65
4.24	q - Quit (Leave the Editor)	66
4.25	r - Read a File	67
4.26	s - Substitute Text	68
4.27	t - Translate a Key on Input	70
4.28	u - Until	72
4.29	v - View Screen Options	74
4.29.1	va - Attribute	74
4.29.2	vc - Center Line	75
4.29.3	vf - Full Display of Text and Attributes	75
4.29.4	vl - Left Margin	75
4.29.5	vr - Right Margin	76
4.29.6	vs - Scroll Screen	76
4.29.7	vt - Set tab settings	76
4.29.8	vz - Zoom the size of your screen	76
4.30	w - Write Buffer to a File	78
4.31	x - Execute a File of Editor Commands	79
4.32	y - Yut?	80
4.33	z - Zap	81
4.33.1	zcc - Zap Cursor Change	82

4.33.2 zcd - Zap Cursor Delete 82
4.33.3 zcD - Zap Cursor Delete Multiple 82
4.33.4 zce - Zap Cursor Erase 82
4.33.5 zcf - Zap Cursor Fill 83
4.33.6 zch - Zap Cursor Horizontal 83
4.33.7 zcl - Zap Cursor Lock 84
4.33.8 zcp - Zap Cursor Purge 85
4.33.9 zcr - Zap Cursor Restore 85
4.33.10 zcR - Zap Cursor Restore Multiple 85
4.33.11 zcs - Zap Cursor Save 85
4.33.12 zh - Zap Home 86
4.33.13 zk - Zap Kopy 86
4.33.14 zlc - Zap Line Center 86
4.33.15 zld - Zap Line Delete 86
4.33.16 zle - Zap Line Erase 87
4.33.17 zlf - Zap Line Fill 87
4.33.18 zlj - Zap Line Join 87
4.33.19 zlo - Zap Line Overstrike 87
4.33.20 zlp - Zap Line Paragraph 87
4.33.21 zlq - Zap Line Query 88
4.33.22 zlr - Zap Line Restore 88
4.33.23 zlR - Zap Line Restore File 88
4.33.24 zls - Zap Line Save 88
4.33.25 zlt - Zap Line Tag 89
4.33.26 zlu - Zap Line Untag 89
4.33.27 zlw - Zap Line Write 89
4.33.28 zm - Zap Message 89
4.33.29 zp - Zap Purge 89
4.33.30 zq - Zap Query 90
4.33.31 zv - Zap Version 90

5. DEFINING YOUR OWN MACROS 91
5.1 What is a Macro 91
5.2 Multi-line Macros 93
5.3 Macros Containing Branches 93
5.4 Suggestions 95

APPENDIX A - ERROR MESSAGES 97

1. INTRODUCTION

ED is a full screen editor for both your console and attached terminals. Quantum has structured this documentation into several major sections.

1. Tutorial Guide	This section consists of a conversational introduction to the Full Screen Editor. It contains examples which should be attempted on your PC as you read. It is highly recommended that all users, regardless of their level of computer experience read this guide. The first several pages contain a complete reference to all the defined function and cursor keys. It will help you correlate the many functions available. Upon completion of the guide you should be capable of performing most editing tasks. For most users, this may be all you need to know about the editor.
2. Using Terminals	This section describes how to configure ED for attached terminals. No configuration is necessary for the console.
3. Reference Manual	This section consists of a detailed description of every command supported by the editor. You will discover that the defined function and cursor keys are in fact implemented as one or more of these commands. Anyone who is going to be using the editor on a regular basis should read the preliminary sections up to the APPEND command. The description of the SUBSTITUTE and GLOBAL commands are also highly recommended. Should you wish to define your own function key operations then it is imperative that you read and understand ALL sections.
4. Defining Macros	This section has been written as a tutorial guide in the writing of macros. It should be sufficient to get you started. However, the best way to learn about macros is to experiment.

© Quantum Software Systems Ltd.

2. TUTORIAL GUIDE

The full screen editor is a program which allows you to type in text, edit it and save it away in a file. The editor itself treats your text as a series of lines consisting of from 0 to 512 characters. It was designed as a program development editor first and a word processor second.

When operating on text from a file, the editor reads a copy of your file into memory (we will call this a buffer). Any changes you make to your file copy (the buffer) will not affect the original file until you issue a write command to save your buffer. Should you change your mind about saving the particular changes you have made, you may exit without saving, or you may reread a copy of the original file.

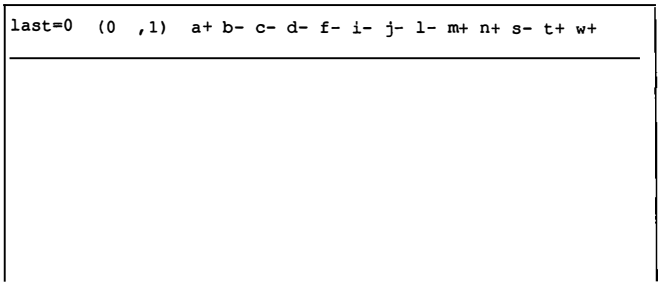
The maximum number of lines you can edit will depend on how long each line is and the amount of available memory space. The maximum number of real characters in a file that may be edited is on the order of 60,000. Larger files must be split and if necessary rejoined after editing.

2.1 Getting Started

Assume you want to enter some text and save it away in a file. Making sure that your commands disk (/cmds) is in one of your drives, enter the command.

ed

The editor will load from disk, clear your screen, then present a screen that should look like this.



The top line of the screen is the status line and provides information about the size of your file, where in the file you are currently located, and what editing options you have selected. This line is kept up to date by the editor. You may NOT enter text into this area.

The second line of the screen is the command line. It is the area where you may type editor *commands* which are to be executed. On the monochrome display, a solid line appears after this line.

The rest of the screen is for text. If your text is longer than or wider than the screen, then this space represents a window into your text. As you progress, you will quickly learn how to position your window to view and/or modify any part of your file.

If you wish to exit from the editor, you can quit by typing a 'q' followed by a carriage return on the command line. If you try this later (after some text has been entered), you may be greeted with a message indicating that you have changed some text but have not saved it away. You can force an exit without saving your text by typing a carriage return to clear the error and a 'qq' followed by a carriage return. We mention quitting here to rescue those people who jump in and then get called away before they can read the rest of the manual.

Now that we have our bearings, let's look a little closer at these three areas.

© Quantum Software Systems Ltd.

2.1.1 The Status Line

This line is composed of three parts. On the left "Last=0" indicates that the last line of your text is line 0. You therefore have an empty buffer (zero lines). Next to this, the numbers in brackets "(0,1)" indicate the current position of the cursor in the file. It is of the form (row, column). The row indicates your current line and for all but an empty buffer it will range in value between 1 and Last. On attached terminals you may find that the column number does not change.

Next on the line are your current editing options. A '+' after an option indicates that the option is ON while a '-' indicates that it is OFF. Some options which are meant to catch your attention when ON will display a *flashing* '+'. These options may be turned on and off at the command line. Some of the options are also tied in with function keys. They are toggled ON/OFF by the indicated key.

Briefly the options have the following meaning:

a - Anchor Alt-a
This option will be better understood after reading the section on pattern matching. For now it is enough to know that the editor has the ability to search for text strings and leave your cursor at the string matched. Should you specify a search for the string "mouse" with this option ON (a+), your cursor will be anchored to the *start* of the pattern matched (in this case 'm'). With (a-), the cursor would be positioned at the character *after* the matched string "mouse" (in this case the character after the 'e').

b - Blank Alt-b
When viewing a piece of text on the screen, lines appear to be padded with blanks to the end of the screen. Since the editor reads files with variable sized lines containing 0 (a line containing only a carriage return) to 256 characters, you may wonder whether the blanks at the end of the line are REAL or not. With (b+), the editor will allow you to differentiate between real blanks by displaying nulls (non-existent characters) as small centered dots.

c - Command Large PLUS key
This option indicates whether your active cursor is in the command area (c+) or text area (c-).

d - Dual Alt-d
This option, like option anchor (a), concerns pattern matching. If OFF (d-) then a search for "mouse" would match the string "MOUSE", "MoUse", MOUsE and so on in your text. If ON (d+), the pattern matcher differentiates between upper and lower case.

f - Fill **Alt-f**
This option when ON (f+) will cause automatic filling of input lines at your defined right margin. The default right margin is column 60. Should you attempt to enter a character in this column then any preceding characters of a word will be moved to the next line. Using this option you may enter text without ever having to type a carriage return to end each line. This is extremely useful when entering documentation and letters.

i - Insert **Ins key**
With (i+), all characters typed will be inserted before the character at the current cursor position.

j - Justify **Alt-j**
This option is used in conjunction with option Fill. When both option fill (f+) and option justify (j+) are ON then when each line is filled it will also be justified.

l - Limit **Alt-l**
This option will flash (l+) when a tagged operation is being limited between a left and right limit.

m- Meta **Alt-m**
With (m+), meta characters are enabled during pattern matching. A meta character is a character which has special significance to the pattern matcher. For instance the character dot (.) is a meta character that will match ANY character, not just a dot. This option will be explained in greater detail later.

n - Newline **F1 or F2 keys**
Whenever a carriage return is entered in the text area while this option is on, a new line will open up after your current line to allow you to type in new text. This can be thought of as line insert mode and is similar to character insert mode (i).

s - Save **Alt-s**
With (s+), your buffer will be automatically saved in the file "autosave" after every 20 lines of input. While writing to the disk you may still continue to type up to 256 characters per line, however, your keys will not be echoed (shown on the screen) until the write is complete.

t - Tabs **Alt-t**
This option is similar to option blank (b). Tab characters in your text are expanded into enough spaces to reach the next tab stop. What may appear as 4 spaces may only be one 'real' character. Turning this option ON (t+) will display the actual tab character as a right triangle. With (b+) also on, the spaces which pad the tab to the next tab stop will be displayed as centered dots

 © Quantum Software Systems Ltd.

since they do not exist within the buffer. Tabs are characters which are heavily used within C programs for indentation.

w- Wrap Alt-w
This option allows pattern searches to wrap around from bottom to top or top to bottom when ON (w+). If OFF then pattern matching will stop when it reaches the last or first line of the buffer.

2.1.2 The Command Line

As previously stated, this line is used for entering text to be interpreted by the editor as commands. If you have just entered the editor you should have a flashing cursor (underline character) on the left margin of your command line. There will also be an inactive cursor (rectangular block) in the text area. Whenever you type a character (excluding cursor and most function keys) it will appear where the active (flashing) cursor is, and the cursor will move to the right.

Now try to enter a few commands to change your options. Most options are enabled by typing

o<*option character*>+

and disabled by typing

o<*option character*>-

First let's turn on option blank by typing the string "ob+" followed by a carriage return (don't enter the double quotes). You can turn it off by typing "ob-<CR>" where <CR> designates a carriage return. Finally, you can toggle it by typing "ob~<CR>". A toggle causes the option to change state. If it was previously OFF it will be turned ON and if it was previously ON it will be turned OFF.

Should you make a typing error you may delete characters by pressing the dark grey back-arrow key or may delete the entire line by pressing the Ctrl and X key simultaneously. Both of these keys work on the line containing the active cursor and are similar to the line editing keys of the command interpreter explained in the QNX Operating System manual.

If you type in an unknown command or do something which causes an error, an error message will be displayed on the command line and held until you type a carriage return to clear it. Try typing in "abc<CR>" to generate an error, then clear it by typing a carriage return.

You should NOTE that you may execute any system command while in the editor by preceding it with an exclamation mark (!). For example

```
!ls
!ed another_file
```

The second example will invoke another copy of the editor. When you leave you will return to this editing session.

2.1.3 Text Area

It is this area where your text is displayed and may be directly edited. As you have seen by setting 'ob+', the text area is currently empty. We are now going to enter some text and follow through some examples to illustrate the function of the various cursor and function keys. Option blank should be ON for these exercises (b+).

2.2 Appending New Text (F1 key)

To enter the text area with the intention of appending new text, you should press the F1 key. The cursor in the text area will become active (flashing) and the command line will be cross-hatched. You will also see option newline enabled (n+) and your last and current line on the line will now be (1,1) not (0,1). The dark back arrow key and Ctrl X keys behave in the same manner as they did on the control line.

Type in the following text, typing carriage return at the end of each line. We apologize to J. R. Tolkien for the misquote.

Three Rings for the Eleven-kings under the sky,
Nine for Mortal Men doomed to die,
One for the Dark Lord on his light throne
In the Land of Mordor where the shadows lie.
In the Land of Mordor where the shadows lie.

After you finish typing you will want to turn off the option newline (n-). This can be done by simply pressing the F1 key again. F1 is a toggle key. Note that option newline is disabled (n-) and the cursor is positioned at the current line or the last line typed. Please refer to Section 2.7 if you wish to leave the editor and/or save your text.

© Quantum Software Systems Ltd.

2.3 Appending or Inserting Lines (F1/F2 keys)

F1 appends *after* the current line and F2 inserts *before* the current line. Now that you have some text, let's experiment with the cursor keys. Using the four arrow keys at the right, you can move the cursor anywhere on your text (note: If the arrow keys generate numbers, type the Scroll Lock key). Your screen will scroll if necessary. You can move off the right of the screen until your status display indicates that you are on column 512. Note that you can't move above the first line, below the last line or to left of the first character.

Using the arrow keys, position yourself on line 2 directly under the "N" of "Nine". Press the F2 key and note that option newline is now on (n+). A new line has opened up between line 1 and line 2 and the cursor points at the beginning of this new line. Press F2 again and notice how the line will disappear if you decide that this is not the place where you want to insert a new line. Press F2 again and type in the following:

Seven for the Dwarf-lords in their halls of stone,

Now press F2 again to turn off option newline (n-) and use the cursors to position yourself on line 5 under the "I" of "In". Type F1 (n+) to append lines after line 5 and type in the following 2 lines:

One Ring to rule them all, One Ring to find them,
One Disk to bring them all and in the darkness bind them

Now type F1 to turn off option newline (n-). The F1 and F2 keys behave identically when leaving newline mode.

2.4 Using the Del and Back-arrow keys

Position yourself about 10 spaces to the right of the word "throne" and type a letter. The editor will immediately lengthen your line with blanks so it can place your character where requested. You can delete these blanks to restore the text by pressing the back-arrow key or positioning yourself just past the "e" of "throne" and holding down the Del key. The Del key deletes the character at the cursor causing the line to shift over.

In general, you will find the back-arrow key useful for correcting new text being entered, while the Del key is useful for editing existing text. As an added advantage, the Del key saves each character it deletes. Position yourself at the start of a line and hold down the Del key. To restore the last deleted characters press the Ctrl and the Ins keys simultaneously. By holding this key down you can restore the last 256 characters deleted in this editor session. The characters need not be restored where they were deleted. They will be restored at the active cursor

wherever it is, even at the command line. This can be a useful method of moving strings of characters.

Change the word "Eleven" to "Elven" in line 1 by moving the cursor to the second "e" and pressing the Del key. Now move to line 7 and position the cursor under the "D" in the word "Disk" and change this word to "Ring" by simply typing over it. Note that by default, you are always in *replace* mode.

2.5 Inserting text using the Ins key

Now that we can enter new text, replace existing characters, delete characters and re-insert deleted characters, it would be nice to complete our capabilities by inserting new characters.

Move to the word "light" in line 4 and position the cursor under the "l". Change this word to "dark" by pressing the Del key 5 times, pressing Ins (i+), typing the word "dark" and then pressing the Ins key again to disable insert mode (i-).

2.6 Other Cursor keys which Simplify Editing

There are several more cursor keys which will simplify movement through your text as follows:

1. Pressing shift and left tab ⇤, will move the cursor to the start of the current line.

2. Pressing the Ctrl and the right tab ⇥, will move the cursor to the end of the current line. Always press the shift or Ctrl key first before the appropriate ⇥ or ⇤. The ⇥ key by itself is actually the tab key and will place a tab character in your text. Tab characters inserted accidentally in your text can be deleted with the Del key.

3. A Ctrl and either the left or right arrow cursor movement keys will step over words quickly. They will always stop at the end of a line containing text. The keys are symmetrical. If you overshoot, simply go in the opposite direction.

4. A Ctrl and either the up or down arrow cursor movement keys will move your cursor up or down four lines.

To test the other cursor keys, a file exceeding the text area must be used. The file /expl/inform (shipped with the QNX operating system on your boot disk) may be used as a source of text for exercise two.

 © Quantum Software Systems Ltd.

2.7 Saving your Text

At any time you may leave the text area and go to the command area by typing the large PLUS (+) key on the right hand side of the keyboard. Note that the right and left cursor keys, the Del and Ins key will operate on the command line. Other cursor keys refer only to the text area. Write your file away by typing:

w filename<CR> - Note that <CR> is the *carriage return* key. Do not type <CR>.

Filename is any valid pathname as explained in the operating system manual. After your file has been saved, the editor positions you back in the text area where you left off. You can now continue with the rest of the exercises or may leave the editor by typing the Large PLUS (+) key again followed by a 'q'. If you modify the file, the 'q' command will not let you quit without saving your changes. If you do not wish to save your changes type 'qq' to force the editor to exit.

2.8 Exercise 2

Position the active cursor at the command line using the Large PLUS (+) key and enter the command:

e /expl/inform

The 'e' command deletes your current buffer and reads the file into the editor creating a new buffer. Note that the editor checks to make sure any current file you are working on has been saved before proceeding. If you get a warning, type carriage return to clear the error and use the 'ee' command to force the editor to load the new file.

ee /expl/inform

Your buffer now contains too many lines to display on your screen and some lines which are too wide to display in their entirety. You have a window of *n* lines by *m* characters into your text. Try using your arrow keys to move about the current screen. If you attempt to leave the screen it will automatically scroll. If you want to move faster remember the Ctrl arrow keys step over words and the Shift ⇤ and Ctrl ⇥ move to the beginning and end of a line.

Two common reference points in a file are its beginning and end. Press the Home key and you will find yourself back at the first line. Press the End key and you will find yourself at the last line of your buffer. A Ctrl and the Home or End key will move you to the first or last line of your current screen respectively. This means that you have a *local* Home and End as well as a *global* Home and End.

To step through the file a page at a time you can use the PgUp and PgDn keys. These keys lock up at the beginning or end of your file buffer. The cursor will be left at your defined center line which defaults to line 3. If you prefer it to be the first line or another line you can change it by entering command mode (the big '+' key) and typing the *view center* command: eg "vc<*number*>" where *number* is between 1 and (screen length - 2)

To set the defined center line to line 1 enter:

vc1

Occasionally you would like to scroll the screen up or down one line without moving the cursor from its current screen line position. This can be accomplished via the Ctrl-PgUp and Ctrl-PgDn keys.

When editing programs, one of the greatest virtues of a screen editor is its ability to give you context via its full screen display. The unmarked 5 key on the numeric keypad is designed to aid you in this respect. If you move the cursor to any line on the screen then press the 5 key (termed the *center* key), your screen will be redisplayed with that line positioned at your defined center line. Try positioning the cursor on one of the lines at the bottom of the screen and press the 5 key.

We have now exhausted the supplied cursor movement keys and it is time to examine the 10 function keys in more detail. Quantum has found that the odd function keys are in general easier to type and where possible has placed the more commonly used editing keys on them.

2.9 More on the F1 and F2 keys

The F1 and F2 keys as we have seen toggle the new line mode. While in newline mode, you may at any time go to the command area and execute any editor command or toggle any of the options. This includes setting option newline OFF instead of using the F1 or F2 keys as toggle switches. This method of entering new lines is different from most editors and much more flexible. You are not locked into an append-only mode.

For example, if you are entering text with option blank off and want to know where all your "real" blanks are, you can zip up to the command area, turn ON option blank (b+) and zip back without leaving the newline mode. This facility allows you to write your file occasionally without exiting from newline mode. The editor always places you where you left off after your file has been written away.

Also, suppose you have 5 lines of text in your buffer and are half way through adding line 6 when you notice a typing error on line 2. You are free to move immediately to line 2 using the cursor movement keys, correct the error, and return again to where you left off on line 6.

In an earlier example (Section 2.3) you inserted a line using the F2 key and turned the newline mode off. You then moved the cursor keys to another position and appended 2 lines using the F1 key. As you will see from the following examples it was not necessary to turn the newline mode off between the two steps. Newline mode has no effect until you type a carriage return.

Try double spacing the first few lines of the /expl/inform file. Move to the first line of the file by pressing the Home key. Press F1 and type a carriage return and then move to the start of the second line using the cursor movement keys. Type another carriage return and continue appending a few blank lines in this manner.

You could also try the following:

1. Appending and inserting text lines in the file.
2. Correcting and inserting characters in the text using the Del and Ins key.
3. Combining steps 2 and 3 without leaving newline mode.

When leaving the newline mode using either the F1 or F2 keys, the editor will delete the last blank line. Typically when appending text users will enter their last line followed by a carriage return. This will open up an unwanted hole which the closing F1 (or F2) conveniently removes. However, for your protection this automatic delete is conditional upon the line being empty.

2.10 Deleting Lines (F3 key)

Your file now has a few blank lines which you may wish to delete. F3 is the line delete key and will delete the current line. Move the cursor to the start of a blank line and press F3. Continue deleting the blank lines in this manner. Now move to a line containing text and press F3. If you wish very hard and then press Ctrl F2 your line will reappear.

The editor stores deleted lines in a delete buffer. The Ctrl F1 will append the last deleted single line *after* the current line and the Ctrl F2 will insert the line *before* the current line. In this case, we inserted the deleted line because the current cursor line moved down one line when the F3 key was used.

If you delete 5 lines you can restore them one by one using Ctrl F1 or F2. They can be deleted in one text area and "undeleted" into another area. In most cases, this can be used as a method of moving lines of text; however, a simpler method of moving blocks of lines using the F7 and F8 keys will be explained later.

2.11 Filling Lines (F4 key)

When in option fill (f+) your lines will be broken on word boundaries based upon your current right margin. If at some time in the future you would like to change your right margin and refill your text you can accomplish this using the F4 function key. It will take the line your cursor is on and all following lines up to:

1. A blank line which is assumed to be a paragraph separator.

2. The end of the file.

whichever occurs first.

A line may be marked as a paragraph start by typing an Alt-p. A paragraph symbol will appear at the END of the line. The Alt-p acts as a toggle allowing you to mark/unmark a line. Filling will stop at each paragraph start, that line will be indented and filling will then continue until one of the above two conditions is reached. The paragraph start is provided as a means to stop filling between two consecutive lines.

If a group of lines have been tagged as described under the F7 key, then only those tagged lines will be filled. Filling will step over each paragraph stop as described above.

Lines are filled between your LEFT and RIGHT margins. If option justify (toggle with Alt-j) is ON (j+), then the text will also be right justified.

2.12 Centering Lines (Ctrl-F4 key)

This key will center the current line between the LEFT and RIGHT margins. Multiple lines may be centered by tagging them as described under the F7 key.

2.13 Splitting and Joining Lines (F5/F6 keys)

Sooner or later you are going to want to split a long line or join two short lines together. The F5 key will split a line at the current cursor position into two lines. The F6 key will join the cursor line and the following line together. In practice, you normally split a line on a space and no longer want the space after the split. The F5 key checks if the character under the cursor is a space and if so, deletes it before the split. On a join, F6 checks to see if the cursor line ends in a space and, if not, appends one before joining.

If you wish to keep a space on a split or suppress the append of a space on a join, then use a Ctrl F5 or a Ctrl F6. These keys make no assumptions; they simply split and join. Note that the operations of split and join are symmetrical. Try splitting and joining several lines to get used to this function.

2.14 Tagging Blocks of Text (F7/F8 keys)

It is often convenient to tag a group of lines, then select an operation to perform on the lines selected. For example, earlier we mentioned a method of deleting a group of lines. Although there are several ways of accomplishing this, the most convenient way is by a tagged delete. The lines to be deleted are tagged and then deleted as a group.

There are two types of tagging.

1. **Line tagging**

2. **Block (line and column) tagging.**

2.14.1 Line Tagging

This is the simpler form and allows you to tag lines in their entirety. The selected operation applies to the whole line. For example, to delete a block of lines you would locate the first line in the series to be deleted and press the F7 key. The line is displayed in reverse-video to indicate that it has been tagged. F7 is another toggle key which sets/removes a tag, hence the line could be "untagged" by pressing F7 again. This is a more complex toggle which is also tied into block tagging, and you should pause about one second before depressing the key again to untag a line.

Now move the cursor and tag the last line to be deleted using F7. You will notice that all of the lines to be deleted in between the tagged lines now show up as inverse-video. When you set two tags, all lines between the two tagged lines are treated as a tagged block. To perform the actual delete, press the F8 key (the tag operation key). This key will prompt you for a command and typing a "d" will delete all the lines as a block. The deleted lines can be restored as a block using the Ctrl Fl or Ctrl F2 (probably a Ctrl F2 since you would want to insert them before the current line).

You will find that if you delete lines one at a time the editor will restore them one at a time and if you delete them as a block they will be restored as a block. This is explained in detail in the Reference Manual.

The tagging of lines may also be used as a method for moving and copying blocks of lines. The F8 key has five functions:

- **d (delete, already explained)**
- **m (move)**
- **k (kopy)**

- **s (save)**
- **p (print lines on $lpt)**

To try the move option, find a small block of text and tag the first and last lines using the F7 key. Move the cursor to another place in the text, press F8 and respond to the prompt with an 'm'. The tagged block of text will be appended after the current cursor line.

The kopy option is analogous to the move option. Tag a block of lines using the F7 keys. Move to the desired location, press F8, enter a 'k' when prompted and the lines will be copied after the current cursor line. The original tagged block of text remains unchanged.

The save option will save the tagged lines into the file '/tmp/*group.member*', where GROUP and MEMBER will be numbers. The saved text may be restored later using the SHIFT F8 key. Selecting a line paste (l) will insert the text before the line your cursor is on. You may save and restore text between different edit sessions and/or consoles.

The F4 (fill) and Ctrl-F4 (center) keys will operate on tagged lines if any have been set. If not they default to the range indicated on the section describing them.

Some things to remember when using tags:

1. You are allowed a maximum of two tags at any one time. If you set a third tag, then it replaces an existing tag.

2. Tag operations apply to all lines between two tagged lines. If only one tag is set, a tag operation will apply to that line only.

3. The order in which the tags are set does not matter.

4. Tags may be removed from your text by:

    (i) performing a tag operation function,
    (ii) using the F7 key as a toggle switch, or
    (iii) typing a Ctrl F7 which removes/resets all tags.

2.14.2 Block Tagging

Line tagging is sufficient for most editing tasks. It falls short in those cases where you wish to operate on a block of text within a line. This occurs most frequently in the preparation of multi-column text.

© Quantum Software Systems Ltd.

For example, if you wished to move the block of C's

```
AAAAAAA BBBBBBB CCCCCCC
AAAAAAA BBBBBBB CCCCCCC
AAAAAAA BBBBBBB CCCCCCC
AAAAAAA BBBBBBB CCCCCCC

DDDDDDD
DDDDDDD
DDDDDDD
DDDDDDD
```

UNDER the block of B's you would need to tag only the C's, not the A's and B's. Enter the text above and position yourself on the first character of the first line of C's. Depress the F7 (tag key) twice in rapid succession. The first depression will tag the line and the second depression will turn on the block tag feature and set a left limit at your cursor. The characters from the first C to the end of your line will be displayed in inverse video.

Now move your cursor to the C in the lower right corner and depress the F7 key again. You need only depress it once. This will tag the last line of C's and set a right limit. The block of C's should now be displayed in inverse video. If you wish, you may adjust the line range being tagged or the limits of the tag by moving your cursor and typing the F7 key. Unlike line tagging, typing F7 again will not remove a block tag.

Now move your cursor under the first B on the line containing the first row of D's and depress the F8 key. You will be queried with a more extensive list of operations then a simple line delete. They are

d - delete
The tagged block will be deleted. Any text to the right will slide over to the left to fill the gap.

e - erase
The tagged block will be replaced by blanks. In the special case where there is no text to the right of the block then the text is simply deleted. This option will maintain column integrity.

k - kopy
The tagged block will be copied to the current cursor location. The original text is unchanged unless the destination overlaps the tagged block.

m - move and erase

This is a combined KOPY followed by an ERASE.

M - move and delete
This is a combined KOPY followed by a DELETE.

s - save
The save option will save the tagged text into the file '/tmp/*group.member*', where *group* and *member* will be numbers. The saved text may be restored later using the SHIFT F8 key. Selecting a column paste will insert the text at your cursor. You may save and restore text between different edit sessions.

p - print
Print tagged block on the printer ($lpt).

Select the 'm' or "M" operation to move the block.

2.14.3 Insert Mode and Block Move and Copy

When performing a tagged block MOVE or COPY with option insert off (i-), the moved text will simply overlay any existing text. If option insert is enabled (i+) then the text will be inserted before the cursor. This can add considerable convenience. For example you may move the block of C's in front of the block of D's by tagging it, enabling option insert, and performing a move to the first character of the first line of D's. You should tag extra spaces to the right so the columns line up. After the move, delete the extra block of C's. It is recommended that you always check the state of option insert when performing a tagged block move.

2.14.4 Re-setting the Last Tagged Lines or Block

The Ctrl-F7 is a toggle which can be used to retag lines or a block. Depressing it with NO tags set will restore the last tags set. Depressing it with tags set will clear all tags.

Tags are set on absolute line and column numbers and inserting or deleting lines and characters may cause your tags to move with respect to your text.

2.15 Re-executing Commands (F9/F10 keys)

These two keys are used for re-executing command line commands. The F9 key will re-execute the last command. F10 will redisplay the last command allowing you to edit it, if needed, before typing a carriage return to execute it. These functions will be more useful as you learn more editor commands in the next sections.

© Quantum Software Systems Ltd.

2.16 Zooming Your Text

If you have an EGA card and have mounted the EGA library

mount lib /config/glib.ega *Must be executed before mounting any consoles*

your console will support 25 by 80 text and 43 by 80 text. Typing Alt z will zoom your display between these two modes. It is possible for custom screen drivers to be written which support more than two screen sizes. In this case the editor is capable of supporting up to 6 different screen sizes.

When the editor exits it will return your screen to the size it had upon entry. Note that outside the editor the STTY command may be used to set your screen size.

stty rows=43
stty rows=25

2.17 Tabs

The editor has tab stops set every four columns with the first tab set on column five. Typing the tab key will enter a tab character at your active cursor. When displayed on your screen the tab character will be expanded into the necessary number of spaces to move to the next tab stop. Try inserting a few tabs into your text. You can display the tab character as a right triangle by turning on option tabs (ot+). By also turning on option blank (ob+), padding spaces will be displayed as small centered dots since they do not exist within your text.

Tabs are not treated with any special significance internally. They only affect your display and cursor movement on the display. You can not position your cursor on the padding spaces following a tab, only on the tab character itself or the first real character following it.

The use of tabs rather than spaces for indentation in structured languages can save considerable typing and file space storage on your diskette.

You can change your tab settings by using the View Tab command. On the command line type:

- **vt2** • **Tabs every 2**
- **vt4** • **Tabs every 4**
- **vt8** • **Tabs every 8**

2.18 Composed Characters

You may enter composed characters directly into your text. These are discussed in the QNX manual in the section on terminal handling. For example, typing:

Alt (release the key) e ' produces an e accent aigu
Alt (release the key) p i produces the symbol pi
etc...

2.19 Line Drawing Characters

You can redefine the Ctrl and Alt cursor keys into line drawing characters by executing a macro file as follows.

x /cmds/box.macros ***Enter this within ED on the command line***

Experiment by typing each Ctrl *cursor-key* and Alt *cursor-key* within the editor. The Home, PgUp, PgDn, End and 5 key should also be used. Some of the keys will move you in the most natural direction for drawing a box, however, the Ctrl down arrow will not move beyond the last line of your file buffer. You can press carriage return to open up room.

2.20 Margins

The editor maintains a left margin which determines the point a carriage return will return to and a right margin which determines the point at which filling will occur.

2.20.1 Moving Your Margins (Shift F1 to F6)

A Shift F1 will set your left margin at your current cursor position and a Shift F2 will set your right margin at your current cursor position. Shift F3 and F4 will march your left margin left or right while a Shift F5 and F6 will march your right margin left or right.

2.20.2 Auto Fill and Your Right Margin

Go to the command area (using the Large (+) key) and enable option fill with the command

of+

or toggle it on with the Alt-f key combination. In the text area, append the fol-

 © Quantum Software Systems Ltd.

lowing ten lines (taken from "The Princess Bride" - William Goldman) as one long line. Do NOT type a carriage return.

The year Buttercup was born, the most beautiful woman in the world was a French scullery maid named Annette. Annette worked in Paris for the Duke and Duchess de Guiche, and it did not escape the Duke's notice that someone extraordinary was polishing the pewter. The Duke's notice did not escape the notice of the Duchess either, who was not very beautiful and not very rich, but plenty smart. The Duchess set about studying Annette and shortly found her adversary's tragic flaw. Chocolate.

You will find that the editor will automatically take any partially typed word and move it to the next line when the cursor column exceeds 60, which is your default right margin. This feature allows you to type continuously without having to closely watch the screen. You can change your right margin using the VIEW command which will be described shortly.

2.20.3 Auto Justify

Go to the command area and enable option justify with the command

oj+

or toggle it on with the Alt-j key combination. In the text area, append the following lines.

Prince Humperdinck was shaped like a barrel. His chest was a great barrel chest, his thighs mighty barrel thighs. He was not tall but he weighed close to 250 pounds, brick hard. He walked like a crab, side to side, and probably if he had wanted to be a ballet dancer, he would have been doomed to a miserable life of endless frustration.

You will find that the editor automatically justifies each line with your right margin when it fills. This option is only active when option fill is enabled.

2.20.4 Indenting and Your Left Margin

A Ctrl-b will begin an indent of four spaces (increase your left margin by four) and a Ctrl-e will end an indent (decrease your left margin by four). Append the following 6 lines, holding down the Ctrl and typing a 'b' or 'e' for <Ctrl-b> and

<ctrl-e>.

The year Buttercup turned ten,
<Ctrl-b>the most beautiful woman lived in Bengal,
<Ctrl-b>the daughter of a successful tea merchant.
This girl's name was Aluthra,
<ctrl-e>and her skin was of a dusky perfection
<Ctrl-e>unseen in India for eighty years.

You should note that your left margin only determines the point you will return to on a carriage return. You may use your cursor keys to move to the left of an indent.

Programmers should NOT use this feature for indentation. The TAB key should be used instead. The above technique results in large quantities of spaces in your files. This will result in larger files and slower compiles.

2.21 Line Flags

2.21.1 Overstrike Flag (Alt-o)

This will flag the end of the current line with a left arrow character. When this line is written, the record separator will be replaced by a carriage return. As a result, when this line is printed, no linefeed will be issued and the following line will *overstrike* this line. This flag is automatically set when reading source lines which terminate in carriage returns rather than record separators. This allows reading and editing the output of DOC files which have underlining and/or boldfacing.

2.21.2 Continuation Flag (Alt-c)

This will flag the end of the current line with a bidirectional arrow character. When this line is written, the record separator will be suppressed. As a result, the next line will be *continued* (joined) with this one. When the editor reads a line of greater than 512 characters it will automatically split the input line and set this flag on the split line. This allows for the editing of files containing very long lines.

2.21.3 Paragraph Flag (Alt-p)

This will flag the end of the current line with a paragraph symbol. This is used by the fill key (F4).

 © Quantum Software Systems Ltd.

2.22 Some Simple Editor Commands.

You should now be able to create new files, edit existing files (e *filename*) and write them away (w *filename*). Up until now you have had little reason to leave text mode to go to command mode to execute editor commands. The next sections will introduce you to some useful editor commands.

2.23 Learn Mode

If you have a sequence of keys which you enter often, you may wish to learn them once and assign them to a single key. Let's assume that you wish to learn the string "Copyright © 1983". The next time you are about to enter it, type a 'Ctrl Minus' sign on the keypad and enter a 'Ctrl a' when prompted for the key to learn. Type in your text then signal the end of learn mode by typing a 'Ctrl Break'. From this point on each time you enter a 'Ctrl a' it will be replaced by the learned input sequence.

It is possible to learn very long and complex sequences consisting of text, cursor movements and commands.

2.24 Absolute Line Positioning

The Home, End, PgUp and PgDn keys allow you a coarse means of moving through a buffer, however, if you want to be in the middle of a thousand line buffer they are very awkward to use. If you go to the command line and type in the number of the line you want to go to (followed by a carriage return to execute) the editor takes this as a command to move your cursor to that line. This gives you the ability to move through the file in absolute terms. For example, if a compiler issues an error for line 458 of a source file then upon reading that source file with the editor you can go right to that line.

2.25 Simple Pattern Matching

When editing a file, you often want to be able to say "Find me an occurrence of this string" so you can work on it without knowing precisely where it is. This is especially true when working from a paper listing in which you know the text string you want to edit, but probably not its line number. In the editor you can find a string simply by enclosing it in slashes on the command line. The command

/son/

will cause a search to be made for the string "son". If option dual is off (od-) then the matching will be case insensitive and /son/ will match "SON", "SoN", "sON" and so on. It will also match a line containing "personal" since it contains an instance of "son".

Searching begins at the character AFTER your cursor, resulting in the editor finding the *next* occurrence of your pattern. If the editor searches down to the end of the buffer without finding your pattern and option wrap is ON (w+) it will continue the search at the first line of the buffer and continue until it reaches and tries your current line from behind. If your pattern is not found, an error will be generated.

Assuming you match a line containing your pattern, then that line will become your current line and your cursor will either point to the first character of the string matched (a+) or the character after this string (a-). The default is option anchor enabled (a+); however, if you are adding commas to the end of words you may prefer to switch to a-.

Enclosing a pattern in question marks instead of slashes will cause a search to be made backwards through your buffer. Therefore

?son?

will search for the first occurrence of the string "son" starting at the character before the cursor. If the editor searches backward to the beginning of the buffer without finding the pattern and option wrap is ON (w+) it will continue searching backwards from the end of the buffer. Again if your pattern is not found an error will be generated.

The editor is careful always to remember the last pattern you specified. Typing

//

will cause a search for the last pattern specified. This can save typing when looking for multiple occurrences of a long pattern. An even faster method would be to use the F9 key to re-execute the last command.

Up until now we have used the words *string* and *pattern* interchangeably. Although the editor can search for simple strings of characters like

/The quick brown fox/

they are a subset of a more powerful pattern matching facility. This facility is enabled by the option metacharacters (om+) when you define a pattern. When enabled the characters ., @, $, ^, *, /, ?, and [have a special meaning. For instance, a '.' is a pattern which matches *any* character. The pattern

/a.c/

would match an occurrence of a string containing an 'a' followed by ANY character followed by a 'c'. The meanings of these characters are explained in

 © Quantum Software Systems Ltd.

detail in the reference manual (section 3). Unless you intend to read this you should turn off option metacharacters (om-) or prefix all special characters with a backslash character in your pattern. When prefixed by a backslash the special characters mentioned above lose their special meaning. A

/cat\./

will only match the string "cat." regardless of the state of option m. If you want to match a backslash you must type two of them. A

/a\\b/

will match the string "a\b". For non-alphanumerics the rule is simply this. If a character is preceded by a '\' then remove the backslash and take that character literally. This allows you to specify a slash in your pattern. A

/total\/number/

will match the string"total/number". The slash is protected and not taken as the pattern delimeter.

For alphanumerics the rule is slightly more involved. If the two characters following the backslash are hexadecimal (0 to 9 or A to F) then the whole sequence is taken as one character which has the hexadecimal value indicated. A

/\07/

is a pattern consisting of the single character whose hexadecimal value is 07. A/\z/ is just a 'z' while a /\bc/ is the character whose hexadecimal value is bc. If this seems complex or confusing you probably don't need this ability. Just remember to type two back slashes to get one and that you can match a slash (or a ? if you scan backwards) by prefixing it with a backslash.

2.26 File I/O Commands

As we have already seen, we can read a file into the editors buffer with the 'e' command

e filename
ee filename

and write out our buffer to a file with the 'w' command.

w filename

Whenever you read a file with the edit command (e) three things actually occur.

1. Your current buffer is purged. However, your delete buffer is kept, allowing you to delete from one file and undelete into another.

2. The filename you specified is memorized.

3. The contents of the file are read into your empty buffer.

In each example we have specified the filename that the command should operate upon. This is not always necessary. Should you omit the filename it will default to the filename of the last file you specified with your 'e' command. Therefore...

e report

followed by a

w

will have the write command write to the file "report". Likewise, an

e

will simply re-read the last file edited. This is often used when you bungle something and want a fresh copy. Note that in this case you will probably have to issue an

ee

command to indicate that your unsaved buffer should be overwritten. For your protection a simple 'e' will not destroy an unsaved buffer.

If you want to check the name of your current file, you can issue the file command.

f

which will display it in the command area. You can change it (or define it) by following the command with a filename.

f file1

This will define your current filename as "file1".

 © Quantum Software Systems Ltd.

It is often useful to read another file into a non-empty buffer after some specified line. This can be accomplished with the 'r' command.

r file2 - reads file2 into the buffer after the current line

The command may be prefixed by a number (or pattern search) to explicitly indicate a particular line.

10r filename - reads file2 after the 10th line in the text buffer.

/end/r filename - reads file2 after the line containing the next occurrence of "end" in the text buffer.

The read ('r') command does not affect the current filename. If you do not specify a filename after 'r' the editor will read another copy of the current file into your text buffer.

It is also possible to prefix the write ('w') command with a line number or range of lines to write.

w filename - all lines are written
33w filename - line 33 only is written
1,10w filename - lines 1,10 are written
#w filename - all tagged lines are written

Finally, you can append to a file by specifying the write append command which is of the same form as the write command.

wa filename

This is useful when you wish to build a new file based upon lines or blocks of lines from your current file (or many files).

1,4w file - initialize file with lines 1 through 4
24,30wa file - append lines 24 through 30
#wa file - append a group of tagged lines
e newfile - read a new file
1,10wa file - append first 10 lines

2.27 The View Command

Users with a colour display can change the colour of the three display areas using the view attribute command.

va<*area*> <*foreground colour*> <*background colour*>

where: *area* -> 1 - Status line
2 - Command line
3 - Text lines
***colour* -> Number between 1 and 15 inclusive**

The colour card has a design characteristic that causes interference (snow, blips etc...) when you write to the screen memory. To avoid this it is necessary to wait for the horizontal retrace signal before writing characters. This has the side affect of slowing down display updates. A number of colour card look-alikes do not suffer from this problem and full speed updating of the display may be restored with the **stty** command system command.

stty type=1 - fast colour card
stty type=2 - slow colour card

Do not use these commands on a monochrome card. It is always type 3.

Users with the improved cards should place this command in their '/config/sys.init' file.

2.28 Executing System Commands

Any QNX command may be executed from within the editor by preceding it with an exclamation mark (!). For example:

```
!ls
!frel junk
!cc test &
!ed another_file
!list this_file &
```

After executing the QNX command, the editor will pause, waiting for the user to type carriage-return before redisplaying the screen. If you type two exclamation marks (!!) the screen clear and pause will be suppressed allowing you to invoke commands via macros in a hidden manner.

 © Quantum Software Systems Ltd.

Note that Ctrl-Z is recognized within the editor and will bring up a new shell allowing you to execute multiple commands. To return to the editor type a Ctrl-D to terminate the new shell.

2.29 Epilogue

At this point you should be able to perform most editing tasks. However, this introduction has hit on only a few of the editor commands and interested users are strongly urged to read the reference manual, especially if they are interested in defining their own function key operations.

© Quantum Software Systems Ltd.

3. Using Ed on a Terminal

3.1 Setting Your Terminal Type

When ED is invoked on an attached terminal it opens a file called '/config/tcap.dbase'. This file contains information entries for different types of terminals. The entry for each terminal contains the output escape sequences necessary to:

1. **Move the cursor**
2. **Change the text attributes to**
   - **inverse**
   - **highlight**
   - **underline**
   - **blink**
   - **colour**
3. **Clear**
   - **the screen**
   - **to end of line**
   - **to end of screen**
4. **Draw lines and boxes**

The entry also contains the input escape sequences sent by any special keys on the terminal's keyboard. These are typically function and arrow keys.

You may query the terminals in the database by typing the TCAP command

tcap list

and you may query your currently set terminal by typing

tcap query

If the query does not agree with your terminal type, you may change it by typing

tset ***terminal_name***

where the terminal name must be one of those listed in the database. If your terminal is not in the database, you will have to read the documentation on TCAP for defining a new terminal.

3.2 Required Terminal Capabilities

Any terminal which is to be used with ED must support the following capabilities.

1. **Direct Cursor Addressing**
2. **Screen Clearing**
3. **Zero width escape sequences**

The last point requires further explanation. On some terminals, an escape sequence to turn on inverse video takes up a character position on the line. As a result your standard 80 column line will be reduced to 78 columns (one character to turn on, and one character to turn off) ED does will NOT work properly on this type of primitive terminal. Escape sequences must NOT take any physical room on the screen. Fortunately, nearly all terminals work this way.

Although not required, the following capabilities are recommended

1. **Clear to end of line.**
2. **Insert and delete line.**
3. **Inverse video (and to a lesser extent underline).**
4. **Up, down, left and right cursor keys.**

The first two will speed up display updates while the third is necessary for displaying tagged blocks of text. If your terminal supports Highlighting or underline, but not inverse video, then ED will attempt to display tagged areas of text using these capabilities. Without cursor keys, the editor may be painful to use.

You may wish to consider a truly QNX compatible terminal described later in this section.

3.3 Screen Output

Ed will adjust its output to conform to your terminal's screen size. Display updates will be determined by the baud rate of your attached terminal. This will be considerably slower than running ED on the console. Non-ascii characters

control : hex 0 to 1F
extended : hex 7E to FF

will be displayed as a question (?) mark. They are saved and manipulated as the characters they really represent. It is only the display which prints them as question marks.

© Quantum Software Systems Ltd.

Note: on attached terminals, the column position is NOT updated as you move your cursor. To force an update you must type the <SHOW> key, which on a PC keyboard, is the center key on the numeric keypad

3.4 Keyboard Input

It is keyboard input which really differentiates using ED on the console from ED on a terminal. The console keyboard is rich in both function and cursor keys. Unfortunately, most terminals are rather limited in the special keys that they provide. To overcome this it is often necessary to enter a two or three character sequence to simulate a single console key. The terminal database defines keys according to their function. For example, the large PLUS key on the console keyboard is called the SELECT key. It is used by most applications to leave some form of input mode and return back to a command state. This key is very seldom found on a terminal, so a default two character sequence of an

ESC CR - select key input sequence

is mapped into the code returned by the console SELECT key on input. The following table contains the default mapping supplied by TCAP for a terminal with NO special keys what-so-ever. The multi-character input sequences will be mapped into the single key codes returned by the console keyboard. Terminals which *do* support special keys may override these default sequences to match those generated by its keys.

Up Arrow - Ctrl u
Down Arrow - Ctrl j or Linefeed
Left Arrow - Ctrl h or Backspace
Right Arrow - Ctrl r

Home - ESC h
End - ESC e
Page up - Ctrl a
Page down - Ctrl b

Insert - Ctrl n
Delete - Ctrl k
Rubout - Delete or Rubout
Erase line - Ctrl x
Select - ESC CR
Cancel - ESC -
Help - ESC ?
Show - ESC s
Tab - Ctrl i or Tab
Tab to begin - ESC TAB b
Tab to end - ESC TAB e

Alternate - ESC a

F1 to F10 - ESC 1 to ESC 0
F11 to F20 - ESC ! to ESC)

NOTE: The following are the translations which are done for the function keys:

TCAP entry		**Key value as seen by ED**
F1 to F10	=	F1 to F10
F11 to F20	=	CNTL_F1 to CNTL_F10
Alternate F1 to F10	=	SHIFT_F1 to SHIFT_F10
Alternate F11 to F20	=	ALT_F1 to ALT_F10

The ALTERNATE escape sequence may prefix any other escape sequence. It may be used to generate the control Arrows, Page up/down, Home and End keys as well as another 20 function keys. These correspond to the SHIFT and ALT function keys on the keyboard.

© Quantum Software Systems Ltd.

There is no provision for executing any of the ALT letter keys such as ALT-b to turn on option blank. You will have to go to the command line and type the option command directly.

ob+ or ob-

You may examine the input escape sequences in effect for your terminal by typing

tcap keys

3.5 QNX Compatible Terminals

There are several terminals on the market which feature complete QNX compatibility. They feature a PC keyboard which generates the same codes as the IBM console, and a 25 line display with the full PC character set and QNX escape sequences. Phone our Technical Support line for the current list of such terminals. A special TCAP entry exists for this type of terminal.

tset qnxt

Phone our Technical Support line for information on which

© Quantum Software Systems Ltd.

4. REFERENCE MANUAL

4.1 The Syntax of Editor Commands

An editor command is of the form:

*<line range>*C*<argument>*"

where: *<line range>* indicates which lines to operate on.
C is a single character indicating a command.
<argument> is command specific information.

4.1.1 Line Range

The line range specifies which lines the command should operate on. It can consist of zero, one or two line addresses. If no range is specified then it will usually default to the current line. The current line is the line your cursor is on in the text area and will typically be updated by each command to reflect the last line it operated on. Check the section on each command for the exact behavior. A *<line range>* is of the form:

<>
or <line>
or <line1>,<line2>
or <line1>;<line2>
or *
or #

where: <line> is the address of a particular line in your text buffer.

The first form in which NO line address is specified will cause the command to choose a default line address. If not stated otherwise, the default will be the current line.

The second form indicates that the command is to operate on the specified line only.

The third form indicates that the command should operate on the range of lines specified.

The fourth form is similar to the third form except that the current line is set to <line1> upon encountering the ';'.

The fifth form is an abbreviation for "1,$" which indicates that the command should be applied to all lines.

The final form indicates that the command should operate on the tagged lines in the buffer. Depending on the number of lines tagged this can result in two forms:

	<line>	**- only one line tagged**
or	**<line1>,<line2>**	**- two lines tagged**

The elements of a <line range> are line addresses <line> and are composed of:

<number>	This is an ordinary line number referring to the <number>th line of the buffer.
$	This special character refers to the last line of the buffer.
.	This refers to the current line of the buffer.
@	This refers to the line occupying the currently defined center line.
&	This refers to the top line of your currently displayed screen.
%	This refers to the current line if you are in the text area and line zero if you are on the command line. It is often used in macros by the ZAP (z) command to operate on the command line.
/<pattern>/	This is the address of the line which contains an instance of the specified pattern. The search for <pattern> will begin at the character after the cursor and will continue to the end of the buffer. If no match has been found by that time, and option wrap is ON (w+), the search will wrap around to the beginning of the buffer and continue looking for <pattern> from line one. If no match is found in the entire buffer then an error will be issued. This line search sets the condition register TRUE if a match is found and FALSE if a match is not found.
?<pattern>?	This serves as the line address of the line which contains an instance of the specified pattern. The search for <pattern> will begin at the character before the cursor and will go backwards through the buffer wrapping around from the first to last line if necessary. If no match is found an error will be issued as above.

© Quantum Software Systems Ltd.

This line search also sets the condition register to TRUE on a match and FALSE on no match.

Each line address above may be combined with other line addresses using the '+' and '-' keys to form expressions. For example.

.-5,.+5	**- will specify the five lines before and after the current line.**
&;.+23	**- will specify all lines on the current screen.**
?begin?,/end/	**- will specify all lines between the lines containing the previous "begin" and the next "end".**

If you specify a line address which is outside the buffer you will get an error and the command will NOT be executed. The special character OR-BAR (|) can be used to limit the preceding line addresses to lie within the buffer (between one and $). This is very useful in defining macros. The '|' operator sets the condition register FALSE if the line address falls outside the buffer and needs to be limited. For example:

&;.+23|	**- is a safer form of the example shown above.**

4.1.2 Command Specification Character

Each major command consists of a single character which has been chosen to reflect its nature. For example, the character 'd' was chosen for the delete command and the character 'w' was chosen for the write command. This character must be in lower case.

If a command line is entered which contains a <line range> but no command

44

then the current line is set to the last line address specified. In this case the cursor will move to line 44.

4.1.3 Right Arguments

Some commands require extra information to specify their operation. For example, the MOVE (m) command requires the specification of the destination line address:

<line1>,<line2>**m**<line3>

Other commands like the ZAP (z) command represent a class of commands which are specified by sub-command characters. For example:

zcd

is a zap cursor delete command. The "cd" are subcommands of the zap command and will not be interpreted as the major commands 'c' and 'd'. This form of sub-command is used by several editor major commands.

4.2 Placing Multiple Commands On A Line

The editor allows you to place more than one command on a line. Each command on the line is executed sequentially from left to right. For example:

ob+ot+

will turn on option blank followed by option tab. The command:

1,4d$d

will delete lines one through four and then delete the last line. Note that the line range only applies to the command that it immediately precedes. Should an error occur on any command then execution will be halted and any following commands will NOT be executed.

Some editor commands consume all characters until the end of the line collecting their right argument. They must therefore be the last command on any line on which they occur. For example

e filename

consumes all characters until the end of the line collecting the filename.

4.3 Special Characters

The Full Screen Editor treats a small number of characters as special in certain situations. These characters are described in the following subsections. The only character you can NOT save in your text is an ascii Null (hexadecimal value 00).

© Quantum Software Systems Ltd.

4.3.1 The Newline Character (hexadecimal 1E)

The newline character in QNX is a record separator (hexadecimal value 1E). Source files separate lines by a single newline character NOT a carriage return and/or linefeed. On input, whenever you enter a carriage return (hexadecimal value 0D) it is mapped into a newline character.

When the editor reads a file it collects characters up until a newline, replaces the newline with a null (hexadecimal value 00) and saves the collected characters as a line in your buffer. The point to note is that the newline is NOT saved. It is stripped on a read and added to the end of each line when the file is written.

In the definition of complex macros containing several lines, the lines may be separated by either a carriage return OR a record separator. The supplied macro file has adopted the convention of using the record separator.

4.3.2 The Null Character (hexadecimal 00)

The Null character (hex 00) is used internally by the editor to delimit strings. It is therefore not possible to save this character in your buffer. Should you attempt to enter this character, the line (text or command) will be truncated at that point.

4.3.3 The Meta Characters @$^&.*[

When option meta-characters is ON (m+), then these characters have a very special meaning when used within patterns (they are special only within patterns). The period '.' for example will match ANY character, not just a period. The meaning of these characters is explained in the section on pattern matching.

4.3.4 The Backslash Character \

The escape character on the command line is the backslash. When it precedes a meta character in a pattern it causes that character to be taken literally. That character loses any special significance it might have normally had.

Following a backslash by two hexadecimal characters in a pattern or translate string results in a single character with the hexadecimal value specified. For example \1E is the single character whose hexadecimal value is 1E (a record separator).

4.3.5 The Tab Character (hexadecimal 09)

When displayed on your screen this character will be expanded into the necessary number of spaces to move to the next tab stop. Tab stops are fixed at every four columns with the first stop set on column five. You can display tabs by turning ON option tabs display (ot+).

Tabs are not treated with any special significance internally. They only affect your display and your cursor movement on the display. You can not position your cursor on the expanded spaces following the tab, only the tab character itself or the real character following it.

4.3.6 The Command Character (hexadecimal FF)

On input, the character with hexadecimal value FF will cause all characters up until the next record separator (newline) to be collected (no echo) in a hidden buffer, then executed as a command. Any current text on the command line is NOT affected. This character is used heavily by the translate command when defining macros for the various cursor and function command keys.

This character is only special on input. You can place a hexadecimal FF character in your text by using the substitute command and a \ff escape.

s/C/\ff/ - replace C with the hexadecimal character FF

4.3.7 The Recall Character (hexadecimal FE)

On input, the character with hexadecimal value FE will recall to the command line the last command typed. This character is used by the F9 and F10 function keys.

You can place this character in your text by using the substitute command as above.

4.3.8 The Keyboard Input Character (hexadecimal FD)

When this character is encountered in a macro, the editor will accept a character from the keyboard. If several characters occur in a row, a maximum of that number of characters will be accepted. Entering a carriage return will always terminate input (skipping any remaining FD's) and the carriage return will be discarded.

4.3.9 The Macro Disable Character (hexadecimal A3)

On input, the character with hexadecimal value A3 will prevent the next character from being expanded should a translate be in effect for it. For example, the Home key has a hexadecimal value of A0, but is translated on input into the three character string:

<command char>1<newline>

If you would like to prevent this expansion (to enter the key's value) then you

should proceed it with the '-' key on the numeric keypad which generates the code for the Macro Disable character. You can of course enter the Macro Disable character itself by typing the '-' key twice.

An alternative method of entering a translated key value would use the compose sequence described in the QNX manual for direct hexadecimal input.

4.4 The Condition Register

The editor maintains a special register called the condition register which is set to TRUE or FALSE by some of the editor commands. This register can be tested by the BRANCH (b) command and the UNTIL (u) command to perform conditional execution of editor commands. These commands are commonly used in macros.

4.5 Delete Buffers

The Editor maintains a buffer for deleted characters and another buffer for deleted lines.

4.5.1 The Character Delete Buffer

The character delete buffer is arranged as a stack 256 characters long. Adding a character to a full buffer will cause the oldest character to be lost. In this manner the most recent 256 characters are kept.

The editor maintains primitive commands for:

- **adding a character to the buffer.**
- **removing the last character which was added to the buffer.**
- **purging the buffer.**

These primitives are provided by subcommands of the ZAP (z) command.

The saving of the last deleted character via the Del key is performed by a macro which saves the character under the cursor in the character delete buffer before deleting it. Likewise, the restoration of a deleted character via the Cntl-Ins key is based upon a macro which inserts the last character placed in the delete buffer before the current cursor position.

4.5.2 The Line Delete Buffer

The editor maintains another buffer in parallel with your text buffer called the line delete buffer. This buffer has the same structure as your text buffer, however, it can not be displayed or directly operated on by the editor's many commands. Each time you delete a line via the DELETE (d) command it is moved from your text

buffer into your line delete buffer. You can restore deleted lines using the special forms of the APPEND (a) and INSERT (i) commands which can move lines from the delete buffer back to your text buffer.

The moving of lines between the two buffers is slightly more complicated than is indicated above and is best explained by an example.

If you were to delete 5 lines, one at a time (say via the F3 key) it would be nice if you could undelete them one at a time so that the last line deleted was the first line restored. This is particularly nice when you delete one line too many and just want to restore the last one, not all of them. Conversely, if you were to delete a group of 100 lines as a block (say via a tagged delete) you do not want to have to restore them one at a time but want them restored as a block as well. The above two scenarios describe, from a user's point of view, the editor's implementation of the line delete buffer. Associated with the buffer is a flag which indicates whether the buffer contains a series of single line deletes or one block delete. To avoid confusion, the editor will purge the line delete buffer before adding in the following circumstances.

- You perform a single line delete and the line buffer contains a block delete.
- You perform a block delete. A block delete always purges the line delete buffer before adding the new block.

Put simply, if you delete lines one at a time, they are undeleted one at a time and if you delete a block of lines they are undeleted as a block. Mixing blocks or types is prevented by purging before adding, if necessary.

When working with a very large buffer it is possible for the editor to run out of memory. When this happens it will purge the delete buffer in an attempt to free up some space. You will be warned of this by a message on the command line which you must clear (like an error) by typing a carriage return. Deleting all lines in a file, then attempting to edit another large file will often generate this message. In this case you have all of the original file in memory in the line delete buffer and are trying to read another large file into the text buffer. They may not both fit!

4.6 Break Handling

The editor will terminate any operation gracefully at the earliest possible moment after the BREAK (Ctrl Scroll Lock) key is typed. As a result of the break, any operation may be incomplete on the range of lines specified for a command, however, no line will be left in a partially modified state. Should you break out of an EDIT (e), READ (r), or WRITE (w) command you may only move a subset of the lines into or out of your buffer to the specified file.

 © Quantum Software Systems Ltd.

After servicing the BREAK the editor will leave you in command state.

4.7 The Pattern Matcher

The editor has a very powerful pattern matching facility which will match the class of patterns known as regular expressions. Patterns are used for line searches and by the GLOBAL (g) and SUBSTITUTE (s) commands. It is the editor's pattern matching facility that gives it flexibility in writing powerful macros. For example, the Ctrl left and right arrow keys are implemented by a pattern which searches for the next or previous word in your text. We will attempt to describe the patterns accepted by the editor in a very rigorous manner. It is assumed that option meta characters is ON (m+) during the definition of your pattern. If it is OFF (m-) then the editor will only recognize the class of patterns represented by (a) and (b) below.

(a) The simplest pattern is a single character. Such a pattern matches the given character in either upper or lower case, unless option dual is ON (d+) to make the Pattern Matcher differentiate between cases.

(b) Patterns arranged adjacently form a single pattern. For example:

/abc/

matches any string "abc".

(c) The character **'^'** specifies a pattern which matches the null string at the beginning of the line. Thus a line search of:

/^charm/

would match the string "charm" at the beginning of a line.

(d) The character **'$'** specifies a pattern which matches the null string at the end of the line. Thus a line search of:

/charm$/

would match the string "charm" at the end of a line.

(e) The character **'.'** specifies a pattern which matches any character. Thus a line search of:

/charm./

would match the string "charm" followed by any character on a line.

(f) Any pattern followed by a **'*'** defines a pattern which matches a string of zero or more occurrences of that pattern. Thus:

/b*/

matches the strings "b", "bb", "bbb", etc. In addition, since '*' patterns will match zero occurrences of a given pattern "/b*/" will also match "". The Pattern Matcher will match a '*' construction with the longest sequence matching the given pattern beginning in a given column; for example, if a line contains the string "bbbbbb", "/b*/" will match all six b's as a unit, not individually.

(g) The construction **"@(<number>)"** is a pattern which matches the null string immediately before the <number>th column on the line. Thus a line search of:

@(10)charm

would match the string "charm" starting in character position ten on the line.

If <number> is zero or the character '.' then this pattern will match the null string before the current cursor position in the text area.

If <number> is the character 't' then this pattern will match the null string before the next TAB stop. This can be used to turn runs of spaces into tabs. See the SUBSTITUTE command.

(h) The construction **"[<string>]"** matches any one character in string and no other. Thus:

/[0123456789]/

is a pattern which will match a single digit. Characters in the square brackets are taken literally, without their special meanings; thus:

/[.]/

will match the character '.' and no other. A sequence of characters may be specified by separating the lower and upper character by a dash (-). For example:

/[a-z0-9]/

is a pattern which will match any letter or any digit. To match a '-' you

© Quantum Software Systems Ltd.

may protect it with the backslash (\) character.

(i) The construction **"[^<string>]"** matches any one character that is NOT in string. Thus:

/[^0-9]/

is a pattern which will match any character that is not a digit.

(j) A null pattern "//" is equivalent to the pattern most recently specified within the editor. This feature is convenient for searching through a file for a particular string. For example, if you are looking for the string "hello" you could specify:

/hello/

the first time and:

//

on subsequent searches. An even quicker method of performing this task would use the F9 key to simply re-execute your line search.

(k) Any character preceded by the backslash character '\' loses its special meaning. Thus:

/\\\^\.\$/

would match the string "\^.$".

4.8 Some Pattern Examples

/hello/	- will match the string "hello" anywhere on a line.
/^hello/	- will match the string "hello" at the start of a line.
/hello$/	- will match the string "hello" at the end of a line.
/^hello$/	- will match a line containing ONLY the string "hello".
/ *$/	- will match all trailing blanks (including zero) on a line.

/^.*$/	- will match all characters (including zero) on a line.
/[0-9][0-9]*/	- will match a number like "3", "862", etc.
/[a-z_][a-z_0-9]*/	- will match an identifier in the language C.
/@(10)[0-9]/	- will match a digit in column ten.
/^$/	- will match an empty line.
/^ *$/	- will match a line containing only blanks.
/^\.[a-z][a-z]*/	- will match a line starting with a period followed by a name (such as a command in the text formatter).

4.9 Editor Commands

The following pages contain an alphabetical list of all editor commands.

© Quantum Software Systems Ltd.

4.10 a - Append After Current Line

Syntax:

<line>a
<line>a <text>
<line>ad

Description:

The first two forms of this command turn ON option newline (n+) and open up a newline after the current line. If the second form with <text> is specified then that text will be placed at the start of the newly opened line. This form is often used within a GLOBAL (g) command to append a string after a set of matched lines. The blank between the 'a' and <text> is required.

The third form appends the last deleted line (or range of lines) from the delete buffer. Option newline is not affected.

The append command must be the last command on a line.

Current Line:

Set to the address of the new line opened up.

Condition Register:

Not affected.

4.11 b - Branch

Syntax:

b<number>
b<number>t
b<number>f

Description:

This command allows you to branch over <number>-1 command lines. The first form branches unconditionally, the second form branches if the condition register is TRUE and the third form branches if the condition register is FALSE. The condition register is not affected by the branch. Branches are often used inside macros defined by the TRANSLATE ™ command and make little sense when executed directly.

If <number> is zero then the current command line is re-executed. If <number> is one, then the rest of the current command line is skipped. If <number> is greater than one, then <number>-1 command lines will be read and discarded. A command line is considered as a string of characters terminated by a record separator (\1E).

The BRANCH command can not be used inside a GLOBAL (g) or UNTIL (u) command.

Current Line:

Not affected.

Condition Register:

Not affected.

© Quantum Software Systems Ltd.

4.12 c - Change Lines

Syntax:

<line range>c
<line range>c <text>

Description:

This command deletes the specified range of lines from the text buffer and places them in the line delete buffer. It then turns ON option newline (n+) and opens up one line for input. If the second form with <text> is specified then that text will be placed at the start of the newly opened line.

The change command is functionally equivalent to a DELETE (d) command followed by an INSERT (i) command.

Current Line:

Set to the address of the new line opened up.

Condition Register:

Not affected.

4.13 d - Delete Lines

Syntax:

<line range>d

Description:

This command deletes the specified range of lines from the text buffer and places them in the delete buffer.

If the <line range> specifies more than one line to be deleted OR the delete buffer contains a previous delete of a range of lines, then the delete buffer is purged before appending the new lines.

If a delete of a single line is requested and the delete buffer is empty or contains lines which have been deleted one by one, then this new line is inserted at the beginning of the delete buffer.

Current Line:

Set to the address of the line after the last line deleted.

Condition Register:

Not affected.

© Quantum Software Systems Ltd.

4.14 e - Edit a New File

Syntax:

```
e
e <filename>
ee
ee <filename>
```

Description:

This command deletes the contents of the current buffer (without placing the lines in the delete buffer) and reads in the lines associated with the specified file. Since the contents of the delete buffer are not affected, you can use it to move lines from one file to another by performing a DELETE (d), EDITING (ee) a new file and then undeleting the lines into the new file (AD command).

The first form reads from the currently defined filename while the second form reads from the indicated filename and makes that the current filename. Both of these will not destroy a non-empty buffer which has been modified. The last two forms will edit regardless of the current state of the buffer.

The current filename can be queried or changed by the FILE (f) command.

The EDIT command must be the last command on a line.

The EDIT command can not be used inside a GLOBAL (g) or UNTIL (u) command.

Current Line:

Set to the first line of the buffer.

Condition Register:

Not affected.

4.15 f - File Query/Set

Syntax:

f
f <filename>

Description:

This command allows you to query or set your current filename which is used by the EDIT (e), READ (r), WRITE (w) and EXECUTE (x) commands when you omit their filename.

The first form of this command displays your current filename on the command line and waits for you to clear it by typing the carriage return key.

The second form sets the current filename to be the filename specified.

The FILE command must be the last command on a line.

Current Line:

Not affected.

Condition Register:

Not affected.

© Quantum Software Systems Ltd.

4.16 g - Global

Syntax:

<line range>g/<pattern>/<more editor commands>
<line range>g^/<pattern>/<more editor commands>

Description:

The GLOBAL command checks each line in the indicated line range for the presence of the indicated <pattern>. The first form marks lines which contain the pattern while the second form marks lines which DO NOT contain the <pattern>. The GLOBAL command then executes the following commands for each marked line, with the current line set at the marked line before executing. If no line range is specified then ALL lines are searched for the <pattern>. For example:

g/^Comment/d

would remove all lines containing the string "Comment" at the beginning of the line. To print them before deleting use:

g/^Comment/pd

To Change all occurrences of the string "minimum" to "maximum" and print the changed lines, issue a:

g/minimum/s//maximum/p

To delete all lines which don't end in the string ".c" issue a:

g^/\.c$/d

To append the line "-------" after each line in the buffer, issue a:

g/^/a -------

To reverse the order of lines in your buffer, issue a:

g/^/.m0

Finally, to print the lines around each line containing the string "function", issue a:

g/function/.-5,.+5p

Current Line:

When the GLOBAL command is finished, the current line has the value it had after the last command executed during the GLOBAL operation.

Condition Register:

When the GLOBAL command is finished, the condition register has the value it had after the last command executed during the GLOBAL operation.

© Quantum Software Systems Ltd.

4.17 i - Insert Before Current Line

Syntax:

<line>i
<line>i <text>
<line>id

Description:

The first two forms of this command turn ON option newline (n+) and open up a new line before the current line. If the second form with <text> is specified then that text will be placed at the start of the newly opened line. This form is often used within a GLOBAL (g) command to append a string before a set of matched lines. The blank between the 'i' and <text> is required.

The third form inserts the last deleted line (or range of lines) from the delete buffer. Option newline is not affected.

The insert command must be the last command on a line.

Current Line:

Set to the address of the new line opened up.

Condition Register:

Not affected.

4.18 j - Join Two Lines

Syntax:

<line>j

Description:

This command joins the indicated line to the line following it. As an interesting example:

uf1j

will attempt to join all lines in the file.

Current Line:

Set to <line>.

Condition Register:

Set if a join occurs. Will always be FALSE on last line.

© Quantum Software Systems Ltd.

4.19 k - Kopy Lines

Syntax:

<line range>k<target line>

Description:

This command *kopies* the indicated range of lines to the line following the specified <target line>. The <target line> may be zero to insert before line one but the <target line> may NOT fall within the source <line range>. The source lines specified by <line range> are left untouched in the buffer.

The 'k' for copy was chosen since the letter 'c' was already in use by the Line Editor and it was felt that its function should be maintained in the Screen Editor for consistency. It's not that we kan't spell.

Current Line:

Set to the first of the kopied lines.

Condition Register:

Not affected.

4.20 l - Learn

Syntax:

l <character>

Description:

All input until a Ctrl-Break will be saved as a macro. From this point, each occurrence of <character> will be replaced by the learned input stream. To learn a key sequence for the F1 key you would enter:

l \81

Current Line:

Not affected

Condition Register:

Not affected

© Quantum Software Systems Ltd.

4.21 m - Move Lines

Syntax:

<line range>m<target line>

Description:

This command moves the indicated range of lines from their current position in the buffer to just after the specified <target line>. The <target line> may be zero to insert before line one but may not fall within the source <line range>.

Current Line:

Set to the first of the moved lines.

Condition Register:

Not affected.

4.22 o - Option Query/Set

Syntax:

o<option character><option modifier>
where: <option character> is a single letter
{a,b,c,d,e,f,i,j,l,m,n,s,t,w}.
<option modifier> is a '+', '-', '~' or '?'

Description:

The OPTION command allows you to change or query your editing options. The option selected is specified by the <option character> and the operation is specified by the <option modifier>. A '+' will turn an option ON, a '-' will turn an option OFF and a '~' will cause the option to toggle. A '?' does not affect the option, but it sets the condition register TRUE if the option is ON and FALSE if the option is OFF. The '+', '-' and '~' do not affect the condition register.

Current Line:

Not affected.

Condition Register:

Set TRUE or FALSE if <option modifier> is a '?'.

Each option is briefly described in a subsection below.

4.22.1 oa - Option Anchor

Whenever a pattern search is made, the editor will leave your cursor at the start (a+) or end (a-) of the pattern matched. Should you specify a search for the string "mouse" with a+ your cursor will be anchored to the 'm' in mouse. With (a-), the cursor would be positioned at the character after the matched string "mouse" (the character after the 'e').

4.22.2 ob - Option Blank

When viewing a piece of text on the screen, lines appear to be padded with blanks to the end of the screen. Since the editor reads files with variable sized lines containing 0 (a line containing only a carriage return) to 512 characters you may wonder whether the blanks at the end of the line are REAL or not. With (b+), the editor will allow you to differentiate between real blanks by displaying nulls (non-existent characters) as small centered dots.

© Quantum Software Systems Ltd.

4.22.3 oc - Option Command

This option indicates whether your active cursor is in the command area (c+) or text area (c-). It is often queried and set within macros. The F1, F2 and large PLUS key are examples of macros which do this.

4.22.4 od - Option Dual

This option, like option anchor (oa) concerns pattern matching. If OFF (d-) then a search for "mouse" would match the string "MOUSE", "MoUse", MOUsE and so on in your text. With (d+) the pattern matcher differentiates between upper and lower case.

4.22.5 oe - Option Environment

This option is handled quite differently from the other options. It does not have an ON or OFF state. Each time you specify an oe+ command, your current options will be saved. They may be restored by an oe- command. The editor only maintains a single level of environment stacking. Multiple oe+ commands simply overwrite previous saves and multiple oe- commands simple restore the last environment saved by an oe+. This command is useful within macros where it may be necessary to temporarily change an option to perform a required function. Only the current state (ON/OFF) off each option is saved.

4.22.6 of - Option Fill

This option when ON (f+) will cause automatic filling of input lines at your defined right margin. The default right margin is column 60. Should you attempt to enter a character in this column, then any preceding characters of a word will be moved to the next line. Using this option you may enter text without having to type a carriage return to end each line. This is extremely useful when entering documentation and letters.

4.22.7 oi - Option Insert

With (i+), all characters typed will be inserted before the cursor.

4.22.8 oj - Option Justify

With (j+), all filled lines will also be justified at your right margin.

4.22.9 ol - Option Limit

This option will flash (l+) when a limited tag has been set.

4.22.10 om - Option Meta Characters

With (m+), meta characters are enabled during pattern matching. The meta characters "@$^&*[." are explained in detail on the section describing the pattern matcher.

4.22.11 on - Option Newline

Whenever a carriage return is entered in the text area while this option is ON (n+), a new line will open up after your current line to allow you to type in new text. This can be thought of as line insert mode and is similar to character insert mode (oi).

4.22.12 os - Option Autosave

With (s+), your buffer will be automatically saved in the file "autosave" after every 20 lines of input. While writing to the disk you may still continue to type up to 256 characters per line, however, your keys will not be echoed until the write is complete.

4.22.13 ot - Option Tabs

This option is similar to option blank (b). Tab characters in your text are expanded into enough spaces to reach the next tab stop. What may appear as 4 spaces may only be one 'real' character. Turning this option ON (t+) will display the actual tab character as a right triangle. With (b+) also on, the spaces which pad the tab to the next tab stop will be displayed as centered dots since they do not exist within the buffer. Tabs are characters which are heavily used within C programs for indentation.

4.22.14 ow - Option Wrap

With (w+) pattern searches will wrap around from top to bottom or bottom to top in your buffer.

© Quantum Software Systems Ltd.

4.23 p - Print Lines

Syntax:

<line range>p
<line range>P

Description:

This command clears your screen and prints the specified lines on your screen. If the number of lines exceeds the size of your screen, the command will pause until you type a character to continue. If the first form is used (small p), then any non-printing characters (less than hex 20 and greater or equal to hex 80) will be expanded into a \hh sequence. This can be useful in finding out the hex values of some of the Function and Cursor keys (proceed them with the '-' on the keypad to put them in your text). This command is nearly always used in conjunction with the GLOBAL command. It allows you to display lines which are not adjacent in your buffer. For example:

g/index/p

will print all lines on your screen that contain the string "index". You may not edit the displayed lines using the cursor keys. The editor has temporarily dropped out of full screen mode.

Current Line:

Set to the last line printed.

Condition Register:

Not affected.

4.24 q - Quit (Leave the Editor)

Syntax:

```
q
qq
```

Description:

This command terminates the current editing session. For your protection the editor will not let you quit via a single 'q' if your buffer has been modified since the last time you saved it. You can force a quit without saving by using the second form 'qq'. The Quit command must be the last command on a line and must not be within a GLOBAL to be recognized.

Current Line:

Not affected.

Condition Register:

Not affected.

© Quantum Software Systems Ltd.

4.25 r - Read a File

Syntax:

<line>r
<line>r <filename>

Description:

This command reads in the contents of a file and appends it immediately following the line <line>. If a <filename> is not given, the current file is used. If <line> is omitted, it will append the file after the current line.

The READ command must be the last command on a line.

The READ command can not be used inside a GLOBAL (g) or UNTIL (u) command.

Current Line:

Set to the address of the first line read from the file.

Condition Register:

Not affected.

4.26 s - Substitute Text

Syntax:

<line range>s<dl><pattern><dl><replacement text><dl>
<line range>s<number><dl><pattern><dl><replacement text><dl>

Description:

This command replaces occurrences of <pattern> with the string <replacement text> in the line range specified. If no line address is specified, substitutions are made on the current line. The pattern and replacement text are delimited by a single character <dl>. It is general practice to use the '/' character for this purpose, however, any other character not appearing in <pattern> or <replacement text> may be used instead.

The first form replaces ALL occurrences of the given pattern on a line while the second form only replaces the <number>th occurrence on the line.

The '&' character has a special meaning (with option m+) when used in the replacement text. It will be replaced by the text of the pattern matched. For example:

s/however/&,/

is equivalent to:

s/however/however,/

This can be extremely useful when matching complex patterns where you may not know the exact text matched. The '&' is NOT a special character in <pattern>, and all the pattern matching meta characters are not special in <replacement text>.

The backslash '\' character may be used in the definition of <pattern> and <replacement text> to escape the meta characters, '&' and the delimiter character <dl> so they lose their special meaning. For example:

s/myfile*/\/yourfile\&/

will replace the string "myfile*" with "/yourfile&". You can also match and replace your text with exact hexadecimal character values using a \hh sequence. One common use is splitting a line by inserting a record separator character. The command:

© Quantum Software Systems Ltd.

s/and further more/&\1e/

will split the current line after the string "and further more". You can not match a record separator character in you text buffer as they are not actually stored, but stripped and added when you read and write your buffer to a file.

The substitute command is often used for making QNX command (ec) files. You first create a file of pathnames using the "files" command.

files p=*.c -v >flist - create a list of my C programs

You would then edit the file "flist" and perform the following substitutions to create a command file to encrypt all your C programs.

```
*s/^.*$/<&>&/      - duplicate names with redirection
*s/..$/.e/          - replace last .c with .e
*s/^/crypt /       - prefix the crypt command in front of
                     all filenames
```

You would now write the file, leave the editor and execute the file.

sh flist

The @™ construct can be used to turn runs of spaces into tabs

```
s/@(t)/\01/        - mark tab stops with a Ctrl-a
s/ *\01/\09/       - replace runs of spaces ending
                     on a tab stop with a TAB
```

Current Line:

Set to the address of the last line of the specified range.

Condition Register:

Set TRUE if a successful substitution takes place, otherwise, it is set FALSE.

4.27 t - Translate a Key on Input

Syntax:

t <character> <replacement text>
t <character>
t ? <character>
T <character> <replacement text>
T <character>
T ? <character>

Description:

This command translates an input character into a string of characters. It is this command which makes the macro definition of the function and cursor keys possible.

The first form translates the indicated character <character> into the string of characters <replacement text> whenever that character is typed. The second form removes any translation in effect.

The third form will display the current translation of the indicated character on the command line allowing you to modify it if desired.

The definition of <character> may be a \hh sequence. If you wish to enter the character to be translated directly (you may not know its hexadecimal value) you should precede the character with the '-' key on the numeric keypad. This suppresses any translation currently in effect for the key. It is recommended that you NOT translate the '-' or '\' key. You should keep in mind that one level of backslash escapes is stripped off during the translate command. This unfortunately means that if you wanted to match a backslash in a substitute command, you would have to type four of them to get one.

t \84 \ffs/\\\\/?/\1e - **You enter this**
\ffs/\\/?/\1e - **This would be saved for the definition of F4. The SUBSTITUTE command would turn the \\ into a single \ when the macro was executed.**

If you use the little 't' command then recursive definitions are allowed. If you use a capital 'T' then a macro within a macro will not be expanded.

© Quantum Software Systems Ltd.

The special character \ff (see section on special characters) should be prefixed to any translation that you wish executed as a command. It causes all characters up to the next record separator \1e to be collected (no echo) in a hidden buffer and then executed as a command. Any current text on the command line is NOT affected. This allows you to define command translations which are independent of where your active cursor is. For example:

t \01 \ff/procedure/\1e

would cause a Ctrl-a to be translated into a scan for the pattern /procedure/. Note that the \1e was necessary to force execution. Without it you would have to type the carriage return key yourself at the keyboard. If we were to omit the \ff, then the text would be entered at the active cursor which would probably be ok if you were on the command line, but unpleasant if you were in the text area.

As a further example, to translate the F1 key into the string "Quantum Software Systems" you would enter the command:

t \81 Quantum Software Sysytems

Note that we did *not* prefix the replacement text with a \ff, nor did we end it with a \1e. Anytime you now type an F1, the string "Quantum Software Systems" will be entered at the active cursor, exactly as if you had typed the characters on the keyboard yourself.

Current Line:

Not affected.

Condition Register:

Not affected.

4.28 u - Until

Syntax:

<line range>u<count> <more editor commands>
<line range>u<condition> <more editor commands>
<line range>u<count><condition> <more editor commands>
<line range>u <more editor commands>

where <condition> is: t - TRUE
f - FALSE

Description:

This command is used to repeat a list of commands a number of times until a given condition is satisfied. The list of commands may contain another UNTIL if desired, however, it may NOT contain a BRANCH (b) command. The space between the <count> or <condition> and <more editor commands> is required.

The first form will repeat the list of commands <count> times.

The second form will set the condition register to the opposite of <condition> and repeat the entire list of commands until the condition register matches <condition> AT THE END OF one of the repetitions of the command list.

The third form will set the condition register as in the second form but will repeat the list until either <count> is reached, or the condition register matches <condition>, whichever occurs first.

The final form will repeat forever, or until you type break (or an error occurs).

In all cases the UNTIL will terminate immediately if any command in the list generates an error. In this case the condition register will be in the state it had before the error. The ERROR will NOT be printed, but absorbed by the UNTIL command. This allows you to prevent a command from issuing an error message by preceding it with a "u1 ". For example:

u1 s/IBM/I.B.M/

will not generate an error if the substitute fails.

If both <count> and <condition> are omitted, then the UNTIL will repeat until an error occurs or you type Ctrl-Break.

 © Quantum Software Systems Ltd.

Current Line:

When the UNTIL command is finished, the current line will have the value it had after the last command executed during the UNTIL operation.

Condition Register:

When the UNTIL command is finished, the condition register will have the value it had after the last command executed during the UNTIL operation.

4.29 v - View Screen Options

Syntax:

va<number> <number> <number>
vc
vc<number>
vf
vl<quantity>
vr<quantity>
vs<quantity>
vt2|4|8
vz
vz*number*

Description:

This command allows you to change some of the parameters associated with your screen. The parameter is specified by the character following the VIEW (v) command. The condition register is not affected by this command.

Current Line:

Not affected.

Condition Register:

Not affected.

Each parameter is briefly described in a subsection below.

4.29.1 va - Attribute

This command allows you to change the attributes (colour) of the three regions of your display. The first number must lie between 1 and 3 and indicates the region.

1 - Status line 2 - Command line 3 - Text lines

The second and third number select the foreground and background colour.

0 - Black 1 - Blue 2 - Green 3 - Cyan
4 - Red 5 - Magenta 6 - Yellow 7 - White

© Quantum Software Systems Ltd.

The foreground colour will be intensified if values greater than 7 are used.

8 - Black 9 - Blue 10 - Green 11 - Cyan
12 - Red 13 - Magenta 14 - Yellow 15 - White

This command is ignored when used with the monochrome display. Users with a colour display may wish to add this command to the end of the file '/cmds/ed.macros'.

4.29.2 vc - Center Line

When used without an argument (form 1 above) this causes your screen to be redrawn with the line your cursor is on positioned at your defined center line. You may redefine your center line by specifying a number (second form above) between 1 and 23. The default macro file sets your center line at 3.

4.29.3 vf - Full Display of Text and Attributes

The editor keeps track of the attributes stored on each line. To increase the speed of screen updates on the console, Ed will not update the attributes when it believes they have not changed. Programs such as QSPELL which modify the screen attributes without the editors knowledge may result in permanent attributes which Ed will not remove. The view full command will force Ed to always update the attributes as well as the text on your screen.

4.29.4 vl - Left Margin

This command defines your left margin. The term <quantity> may be a simple number in which case your margin will be set to that value, or it may be a number preceded by a '+' or '-' in which case that value will be added or subtracted from the current value. This lets you define absolute or relative changes to your margin. For example:

vl+5

will increase your left margin by five and:

vl5

will define your left margin to be five. As a special case, a '.' will set your margin to the column that your text cursor is currently on.

Your left margin defines the point you will return to in your text whenever you type a carriage return. The characters before your margin will be filled by blanks unless you have changed your default fill character (see ZAP (z) command).

Your left margin is constrained to lie between 1 and your right margin.

4.29.5 vr - Right Margin

This command defines your right margin in the same manner as "vl" defined your left margin. Your right margin defines the point at which filling (automatic generation of a carriage return) will occur on text entry if you have option fill ON (f+).

Your right margin is constrained to lie between your left margin and column 1000.

4.29.6 vs - Scroll Screen

This command causes your screen to be scrolled. If you specify a negative quantity then the scroll will be backward and if you specify a positive quantity then the scroll will be forward. This command always causes an immediate refresh of your screen. This allows you to look at snap shots of your screen during the execution of an until (or any list of commands). The command:

u20 s/^/+/

would only display the final version of the line you modified when the editor finished execution. The command:

u20 s/^/+/vs0

would give you a snap shot after each iteration of the substitute command.

4.29.7 vt - Set tab settings

You may set your tab stops at every 2, 4 or 8 characters on the screen. The default is 4.

4.29.8 vz - Zoom the size of your screen

When given without arguments this command will switch to the next hardware supported screen size. If the number is given it will switch to

vz0 - Size of screen upon entry.
vz1 - Screen size 1 (25 by 80 on an EGA).
vz2 - Screen size 2 (43 by 80 on an EGA).

vz*X* Hardware dependent.

4.30 w - Write Buffer to a File

Syntax:

<line range>w
<line range>w <filename>
<line range>wa
<line range>wa <filename>

Description:

This command writes out the specified lines to a file. If a <filename> is not given the current file is used. If the <line range> is omitted then ALL lines are written.

The last two forms (wa) will append the specified lines to the end of the file.

If a <filename> is specified and the current filename is not defined, then the specified <filename> will become the current filename. Note that <filename> can of course be a device like '$lpt'.

Current Line:

Set to the address of the first line written.

Condition Register:

Not affected.

© Quantum Software Systems Ltd.

4.31 x - Execute a File of Editor Commands

Syntax:

x
x <filename>

Description:

This command will open up a file and execute its contents as a series of editor commands. If a <filename> is not given the current file is used.

This command is commonly used to load macros via a file of translates. The editor does an EXECUTE on the file "/cmds/ed.macros" immediately after execution begins. An execute file may NOT contain another execute.

The EXECUTE command can not be used inside a GLOBAL (g) or UNTIL (u) command.

Current Line:

The current line is set by the commands in the execute file.

Condition Register:

The condition register is set by the commands in the execute file.

4.32 y - Yut?

Syntax:

<line range>y"<prompt text>"<command chars>"<arguments>

Description:

This command does not know what it is. It will print <prompt text> on the command line then wait for you to type the letter of a command. If the character you type matches the *n*th character of <command chars> then control will transfer to the *n*th line. Each line is separated with a record separator (\1E).

y"k for kopy or m for move"km"\1e\ff#k.b2\1e\ff#m.\1e

would prompt:

k for kopy or m for move

on the command line and pause for you to type a character. If you typed an 'm' then the command:

#m.

would be executed, which happens to be a tagged move to the current line.

If the typed character is not found, then the last command will be executed. The 'n' command is an invalid command and can be used to generate errors on invalid keys. In the following example, a space or any invalid key will generate an error:

y"k for kopy or m for move"km "\1e\ff#k.b3\1e\ff#m.b2\1e\ffn\1e

Current Line:

Set according to the command executed.

Condition Register:

Set according to the command executed.

© Quantum Software Systems Ltd.

4.33 z - Zap

Syntax:

<line range>zcc<character>
<line range>zcd
<line range>zcD<location>
<line range>zce<location>
<line range>zcf<character>
<line range>zch<location>
<line range>zcl
<line range>zcp
<line range>zcr
<line range>zcR<number>
<line range>zcs
<line range>zh
<line range>zk<line>
<line range>zlc
<line range>zld
<line range>zle
<line range>zlf
<line range>zlj
<line range>zlo
<line range>zlp
<line range>zlq
<line range>zlr
<line range>zlR
<line range>zlR"filename"
<line range>zls
<line range>zlt
<line range>zlu
<line range>zlw
<line range>zlw"filename"
<line range>zm<text>
<line range>zp
<line range>zq#
<line range>zq!
<line range>zv<digit>

Description:

This command consists of a number of useful subcommands which relate directly to the capabilities of a full screen editor. The subgroup starting with 'zc' (ZAP CURSOR) deal with the cursor and are further subdivided by a third character to indicate a particular function. The <line range> for the "zc" group

may be a single line address of zero in which case the command will be applied to the command line instead of the text area. Each ZAP function is described in a subsection below.

Current Line:

In all cases, it is set to the last line of the specified range, however, not all ZAP commands use the line range information (zp for example).

4.33.1 zcc - Zap Cursor Change

This function changes the character under the cursor to the character specified. The <character> may NOT be a \hh escape sequence but may be a carriage return. If option insert is ON (i+) then <character> is inserted BEFORE the cursor.

Condition Register:

Not affected.

4.33.2 zcd - Zap Cursor Delete

This function deletes the character under the cursor causing the rest of the line to shift to the left.

Condition Register:

Trying to delete a non-existent character (you are to the right of the end of the line) will set the condition register FALSE. A successful delete sets it TRUE.

4.33.3 zcD - Zap Cursor Delete Multiple

This function will delete all characters from the current cursor position to the position specified by <location>. The <location> is specified in the same manner as the 'zch' command.

Condition Register:

Trying to delete a non-existent character (you are to the right of the end of the line) will set the condition register FALSE. A successful delete sets it TRUE.

4.33.4 zce - Zap Cursor Erase

This function will erase all characters from the current cursor position to the position specified by <location>. This allows erase to end of field (right margin) to be implemented via a

 © Quantum Software Systems Ltd.

The <location> is specified in the same manner as the 'zch' command. If the field does not end the line, the field is replaced with blanks instead of being deleted. This preserves column integrity.

Condition Register:

Set TRUE if any characters are erased.

4.33.5 zcf - Zap Cursor Fill

This function defines the fill character to be used whenever you type a character to the right of your text (indicated by a small centered dot in option blank (b+)). The default is to extend the line with blanks. However it may be changed to any character, a period being a typical example. The <character> may NOT be a \hh escape sequence.

The fill character is often changed to a TAB when used in conjunction with the Ctrl-b (begin indent) and Ctrl-e (end indent) keys when writing programs. Indenting by TABS rather than spaces is strongly recommended.

Condition Register:

NOT affected.

4.33.6 zch - Zap Cursor Horizontal

This function moves the cursor horizontally on the line. The movement may be absolute, relative or based upon a pattern. The field <location> determines the type of movement and can be one of:

<number>	Absolute movement to indicated character <number>.
+<number>	Relative movement forward <number> characters.
-<number>	Relative movement backward <number> characters.
$	Move to end of line.
l	Move to left margin.
r	Move to right margin.
/pattern/	Starting at the current cursor attempt to match the indicated pattern. Note that the line is assumed to start at the cursor and all characters to the left of the cursor are ignored. The pattern /^./ would match the character at the cursor NOT the beginning of the line. Option anchor will determine the new location of the

cursor on a match.

?pattern? Starting at the current cursor and scanning BACKWARDS, attempt to match the indicated pattern. Note that the line is assumed to start at the cursor and extend backwards to the beginning of the line. All characters to the right of the cursor are ignored. Due to the backward scan, the line will appear reversed to the pattern. To match the string "IBM" you would specify the pattern ?MBI?. This is only true of backward searches using the ZCH command.

In all cases above, the cursor is limited to lie between index 1 and 512.

Condition Register:

An attempt to move the cursor outside the text on the line (or a pattern match fail) will set the condition register FALSE, otherwise, it is set TRUE.

4.33.7 zcl - Zap Cursor Lock

When the editor is forced to move the cursor to a line which is not within the currently displayed screen, it will by default redisplay with the cursor line positioned at your defined center line (see vc<number>). The Zap Cursor Lock command, LOCKS the cursor to the TOP or BOTTOM of the screen on a redisplay. Conceptually, if your cursor must move off the top of your screen then, it will remain locked at that position and your text will scroll down as required. Likewise, if your cursor must move off the bottom of your screen, then it will remain locked at that position and your text will scroll up as required.

As an example the UP ARROW and DOWN ARROW are implemented as:

.-1|zcl and .+1|zcl

to prevent a re-centering of your display when your cursor attempts to leave the screen.

Condition Register:

NOT affected.

© Quantum Software Systems Ltd.

4.33.8 zcp - Zap Cursor Purge

This command purges all characters in the character delete buffer.

Condition Register:

NOT affected.

4.33.9 zcr - Zap Cursor Restore

This command causes the last character put in the delete buffer to be restored at the current cursor position. If option insert is ON (i+) the character will be inserted before the cursor.

Condition Register:

Set TRUE if the character delete buffer contains at least one character to restore.

4.33.10 zcR - Zap Cursor Restore Multiple

This command will restore the last <number> characters put in the delete buffer at the current cursor position. If option insert is ON (o+) the characters will be inserted before the cursor.

Condition Register:

NOT affected.

4.33.11 zcs - Zap Cursor Save

If the cursor is positioned on a character, then it is saved in the character delete buffer. If the cursor is not on a character (off to the right of the line) then nothing is saved.

The character delete buffer may hold a maximum of 256 characters. Saving more than this number causes any new character to be inserted at the front and the oldest character is lost. Only the last 256 characters are kept.

Condition Register:

Set TRUE if a character is saved.

4.33.12 zh - Zap Home

This command will move your cursor to the top left hand corner of a tagged block of text. For example, say you wish to write a macro which saves a tagged block of text to a file, executes the SORT command and replaces the tagged tagged text when sorted. It is necessary to ensure that the cursor is positioned at the top left corner of the text when you read the file back in.

```
t\01 \ffzh\1e\ffzp#zlw"tmp"!!sort tmp +r\1e\ffzpzlR"tmp"\1e
```

Condition Register:

NOT affected.

4.33.13 zk - Zap Kopy

This command provides a convenient method of kopying a text line to the command line and vise versa. A line address of zero designates the command line. For example

0zk. - kopy command line to current line
.zk0 - kopy current line to command line

Condition Register:

Not affected.

4.33.14 zlc - Zap Line Center

This command will center the text between the left and right margins of the indicated lines.

Condition Register:

NOT affected.

4.33.15 zld - Zap Line Delete

This command will delete the text between the left and right margins of the indicated lines.

© Quantum Software Systems Ltd.

Condition Register:

NOT affected.

4.33.16 zle - Zap Line Erase

This command will erase the text between the left and right margins of the indicated lines.

Condition Register:

NOT affected.

4.33.17 zlf - Zap Line Fill

This command will fill the text between the left and right margins of the indicated lines.

Condition Register:

NOT affected.

4.33.18 zlj - Zap Line Join

This command will set the join flag on the indicated lines.

Condition Register:

NOT affected.

4.33.19 zlo - Zap Line Overstrike

This command will set the overstrike flag on the indicated lines.

Condition Register:

NOT affected.

4.33.20 zlp - Zap Line Paragraph

This command will set the paragraph flag on the indicated lines.

Condition Register:

NOT affected.

4.33.21 zlq - Zap Line Query

This command will set the condition code true if there are any marks set.

Condition Register:

TRUE if any marks set.

4.33.22 zlr - Zap Line Restore

This command will restore the contents of the delete buffer where the current text cursor is. The delete buffer is not affected.

Condition Register:

NOT affected.

4.33.23 zlR - Zap Line Restore File

This command will read the contents of a file into the delete buffer then restore the contents of the delete buffer where the current text cursor is. If no file name is specified, then the file '/tmp/group.member' will be used where group and member are numbers.

Condition Register:

NOT affected.

4.33.24 zls - Zap Line Save

This command will save the text between the left and right limit at the end of the delete buffer. Limits are set by tags.

Condition Register:

NOT affected.

© Quantum Software Systems Ltd.

4.33.25 zlt - Zap Line Tag

This command TAGS the indicated line if it was untagged and UNTAGS it if it was TAGGED. If there are already two tags in effect then this tag will replace one of the tags. NOTE: If two 'zlt's occur within 1/2 second then option limit will be enabled and the line will not be untagged.

Condition Register:

NOT affected.

4.33.26 zlu - Zap Line Untag

This command removes all tags in the file if any are set or sets the last tags if there are not any tags set. It is a toggle.

Condition Register:

NOT affected.

4.33.27 zlw - Zap Line Write

This command will save the text between the left and right limit at the end of the delete buffer. Limits are set by tags. The data is then saved into a file. If no file name is specified, then the file '/tmp/group.member' will be used where group and member are numbers.

Condition Register:

NOT affected.

4.33.28 zm - Zap Message

This command will print the message which follows on the command line. This is useful as a prompt.

Condition Register:

NOT affected.

4.33.29 zp - Zap Purge

This command purges all lines in the line delete buffer.

Condition Register:

NOT affected.

4.33.30 zq - Zap Query

This command sets the condition code based upon the character which follows the zq.

Condition Register:

- TRUE if any tags have been set
! - TRUE if zero exit status of last command

4.33.31 zv - Zap Version

This command sets the condition code based upon the version number of the editor. The version is a number between 0 and 9. If the editors version is greater than or equal to the digit specified the condition code will be set TRUE.

zv1 ***Set condition code TRUE if editor version 1 or later***

Condition Register:

TRUE if editor is greater than or equal to indicated version.

© Quantum Software Systems Ltd.

5. DEFINING YOUR OWN MACROS

This section is intended for the advanced user who wishes to add functions to the supplied macro file "/cmds/ed.macros" or define a set of new macros for a particular editing task at hand. Before trying your hand at new macros you should have read both the Tutorial Guide and Reference Manual in detail. Macros are basically very simple to write, BUT only if you have a good grasp of the many editor commands described in the reference manual.

5.1 What is a Macro

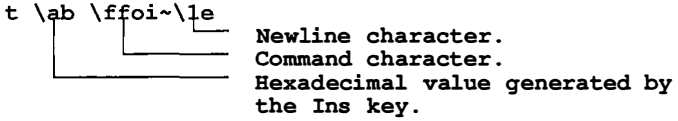
A macro is just the simple replacement of an input key with a string of characters. The replacement string contains the keys you would have to type to perform the required function manually. As an example, let's look at the definition of the Ins key. Each time it is typed, it toggles option character insert (oi). You could perform this task manually by going to the command line and typing the command

oi~

which will toggle the option. You would then want to go back to the text area if that's where you came from. If you were already on the command line with a partially typed command then the situation is slightly more complex. You can't type in a new command on the command line without deleting what is currently there. To overcome this difficulty, the editor provides a very special character called the <command char> which causes all input following it, up to the next newline to be collected (without echo) in a hidden command buffer which is then executed. By prefixing all commands with this character you can execute commands regardless of whether you are on the command line or in the text area. The value of the command character is hexadecimal FF and may be entered directly from your key board by holding down the Alt key and typing the backarrow key normally used for character delete. You can now toggle insert mode at ANY time by typing the string

<command char>oi~<newline>

where <command char> is an Alt-backarrow and <newline> is either a carriage return or a record separator (Ctrl ^). To turn this into a macro for the Ins key you would translate the value generated by the Ins key into the above string.



If you leave off the command char then the string

oi~

followed by a newline would be entered at the active cursor. Likewise, if you leave off the newline, then the command will be collected, but you will have to supply the newline yourself by typing a carriage return.

Most macros are this simple. However, care should always be taken to ensure reasonable behavior when a requested operation may fail. An example of this is the down arrow key on the keypad. It is designed to move you down one line. This could be performed by executing the command

.+1

which works fine until you try to move past the last line in your buffer where you will be greeted by an unpleasant error message which will quickly annoy any user. The simple solution is to limit the line address to remain within the buffer using the 'l' operator. The resulting macro definition would be:

\ff.+1|\1e

which behaves in a friendlier fashion. As one final point, to prevent your screen from being re-centered when you attempt to leave the screen, you might add a ZAP CURSOR LOCK command resulting in

\ff.+1|zcl\1e

Lets look at one more simple example illustrating the importance of planning your macro's behavior in all possible situations. You want the PgDn key to display the next page of your buffer. Your text area is 23 lines so you might simply attempt to move forward 23 lines via a

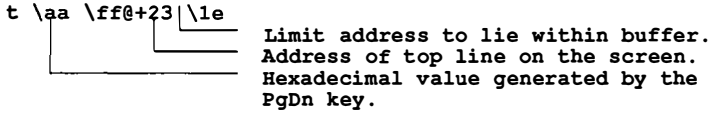
.+23

This has two problems

1. You may not have another 23 lines in the buffer.

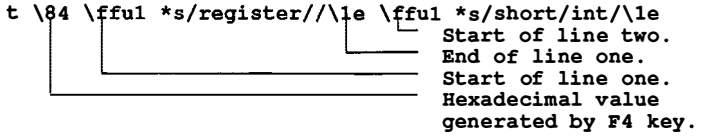
2. If your current line was not on line 1, then you would miss part of the next page. You should be jumping forward relative to the address of the top line of the current screen. The PgDn is currently defined as

 © Quantum Software Systems Ltd.



5.2 Multi-line Macros

Most of your macros will consist of one or more editor commands entered as a string which makes up a single command line. There are cases, however, where you will find that you need to enter a multi-line command. You may do this by simply embedding <newline> characters in your translate string. The macro

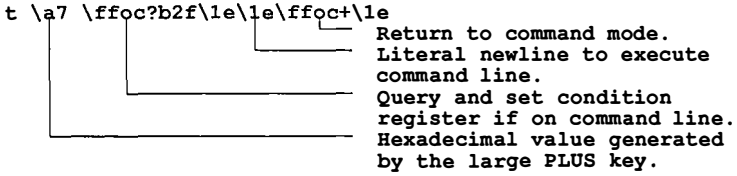


defines a macro which will remove all occurrences of the string "register" and change all occurrences of the string "short" to the string "int". The UNTIL one (u1) will prevent an error from being printed if the substitute fails to find a match. By placing them on two lines we are guaranteed to perform the second substitute even if the first one fails. Remember, an error (even inside an UNTIL) terminates execution of the current line.

5.3 Macros Containing Branches

The most common use of multi-line macros involves the BRANCH (b) command. This command was included in the editor to allow you to perform conditional execution of editor commands within macros. A simple example is the large PLUS key on your keypad which performs one of two tasks. If you are in the text area it simply places you on the command line. If, however, you are already on the command line, it simulates your entering a carriage return to execute any command, then places you back on the command line. Hitting carriage return would normally execute any command then put you back in the text area.

This key is defined as.



This macro is interesting for two reasons.

1. It makes use of the BRANCH (b) command to conditionally execute editor commands. If you are in the text area it will skip two <newline> characters and only execute the "\ffoc+\1e" which will place you in command mode.

2. It mixes literal text (which is entered at the current cursor) with command text (which is preceded by a \ff and is collected in an execute buffer). If you were already on the command line it will not branch, and therefore enter a <newline> on the command line which will cause it to be executed. You will then fall into the code which sets option command and goes to the command line.

Defining complex macros which contain several branches and several lines quickly becomes confusing when displayed as a single line with embedded <newline> character escapes. The macro for the F2 key illustrates this.

t \82 \ffon?b3t\1e\ffi\1e\ffb4\1e\ffon-zch1\1e\ffb2t\1e\ffd\1e

This macro can be better understood when displayed as the multi-line sequence

t \82 \ffon?b3t	**- Check if option newline is on or off.**
\ffi	**- If it is off, then do an insert command**
\ffb4	**and skip the rest of the macro.**
\ffon-zch1	**- Else go to the beginning of the line and skip**
\ffb2t	**to the end of the macro if line isn't empty.**
\ffd	**- Delete the empty line.**

It should be remembered that the zap cursor horizontal command (zch) will set the condition register false if you move to a position which does not contain a character. If there is no character in the first column then the line must be empty.

There is a macro which will allow you to type in a macro in your text buffer in the multi-line form above and convert it into a single string on the command line which you may type carriage return to enter. This macro is defined in your

 © Quantum Software Systems Ltd.

"/cmds/ed.macros" file, but has been commented out by a double quote. You may remove this quote and then execute the file

x /cmds/ed.macros

to define this macro as your Alt-m key. If you type the Alt-m key in the text area, it will compress your buffer into a single line and move it to the command line for execution. If you are on the command line, then it will take a macro which you have displayed via the

t ? \hh

command and place it in your text buffer in a multi-line format.
For interest this macro is defined as:

```
t \84 \ffoc?b7f*da
\ffon-om+0zk1
\ffu1 s/\\\\\\\\/\\80/
\ffu1 s/\\\1e/\\1e/
\ffu1 *s/\\80/\\\\\\\\/
\ff0zce255 1zch1b5
\ffom+                  - Start of code in the case
\ffu1 *s/$/\\\\1e/        where you are in the text area.
\ffu40 1j
\ff1zk0zch1oc+
```

This macro uses UNTIL commands to prevent errors should a substitute fail.

In the case where you are on the command line, it deletes your buffer and places an empty line in it via the APPEND command. It then copies the command line to the empty line and splits the line at each \1e. Note that it is not tricked by a \\1e sequence.

The reverse process appends each line with a \1e sequence, joins all lines, then copies it to the command line. It leaves the joined line in your text buffer allowing you to save it with a write or write append command.

5.4 Suggestions

At this point you should examine the macros defined in the file "/cmds/ed.macros" to gain further insight into the writing of new macros. You may examine them by reading the file, or by displaying each key's translation one at a time via the "t ? \hh" command and then using the F4 key to show it as a nice multi-line sequence. If you do not know the hexadecimal value of a key, then you may enter it's literal value by preceding it with the macro disable key (MINUS key on the keypad). For

example, you could type either

t ? \81
or
t ? <MINUS key><F1 key>

The possibilities for macros are endless. Quantum encourages their fullest possible use, but warns you that creating them is very much a black art...

© Quantum Software Systems Ltd.

APPENDIX A - ERROR MESSAGES

The Full Screen Editor prints any errors it detects on the command line and pauses for you to type a carriage return to clear the error.

out of memory

The Editor was unable to allocate space to replace an existing line or add a new line.

unknown command

The editor encountered an unknown major command or subcommand character.

invalid pattern specification

A pattern was specified which was not acceptable because:
- Terminating delimiter of pattern missing.
- Pattern exceeds 128 characters.
- Pattern starts with a '*'.
- // pattern specified when no pattern was defined.

buffer has been modified, use qq to quit without saving

An attempt to leave the editor via the quit command when your text buffer has been modified since your last write to a file.

invalid line number or line range

A command has detected an invalid line number or range of line numbers. This can be caused by:
- <line1>,<line2> where <line1> is greater than <line2>.
- <line> is greater than the last line in the buffer.
- <line> is zero on a command other than a,k,m,r or z.
- Target of a move or kopy falls within the source range.

x command encountered within an execute file

An attempt has been made to nest execute commands.

current file not defined

An e,r,w or x command was entered without a filename and no current filename was defined. You can define it with the f command.

unable to access file

The file system was unable to open the requested file. This can be caused by:

- Invalid filename specified.
- No permissions to open file. (read, write or append)
- Filename specified can not be found. Check spelling.

syntax error

A recognized command was specified incorrectly. Some commands which do not take a line range will flag a syntax error if given one (eg: e, f).

filename too long

A pathname of more than 64 characters was specified on a e,f,r,w or x command.

unknown option

An unknown option character was specified in the option command.

line greater than 512 chars

On a read from a file, a line greater than 512 characters was encountered. The line is broken at 512 characters. Reading will continue when you type carriage return. The j and s commands can also cause this error.

© Quantum Software Systems Ltd.

pattern not found

The pattern specified in a line search or a substitute command was not found anywhere in the buffer.

disk error

An error has occurred while trying to read/write a disk file. Make sure that the disk is not write protected (if writing). Otherwise, this error may signal a defective diskette or a bad block.

Attempt to write to a file which is not the current file. Use ww to force.

You are trying to write to a named file which is different from the one you edited. Perhaps you wanted to type "e file" and missed the 'e' and hit the 'w' instead. If you are sure, recall the line using the F10 key and insert a second 'w' for a command of "ww file".