7 Input/Output Utilisation

7.1 Introduction

Chapter 6 has already introduced the concepts and procedures for allocating I/O channels. In order to make use of allocated channels, it is necessary to analyse the sending of data to, and reception of data from a channel.

The traps dealing with channel utilisation can be split up into three fairly well defined groups. These three groups are summarised in the following three subsections.

7.1.1 Serial I/O Calls

These deal with fully redirectable I/O within the QL. Serial I/O operations include communication with the keyboard, RS232 serial lines, screen, printer, microdrives etc. (subject of course to the read/write restrictions of particular devices). Only the number of the channel to which data is to be sent, or from which data is to be received needs to be specified. All other information about the particular characteristics of a file will already have been provided by the OPENing process (see chapter 6).

When dealing with the fully redirectable I/O system, there are three possible return states. Return can be inhibited until the specified process is complete, forced to occur immediately (ie. with no wait), or programmed to occur after a pre-defined time (or completion of the process if earlier). The timeout period is specified as either zero, a time between 0 and 10 minutes or indefinite. The time increment is as an integral number of 50 or 60Hz cycles.

Serial I/O call summary

D0	Name	Description
00	IO.PEND	Checks for pending input
01	IO.FBYTE	Fetches a byte
02	IO.FLINE	Fetches a line of characters terminated by <LF>
03	IO.FSTRG	Fetches a string of bytes
04	IO.EDLIN	Edits a line of characters (console driver only)
05	IO.SBYTE	Sends a byte
07	IO.SSTRG	Sends a string of bytes

7.1.2 Screen output control

Control of the video display is not in itself part of the redirectable I/O system. However, the screen is an integral part of the QL and as such must be included under I/O utilisation. By screen output control, such processes as modifying the window, controlling the cursor, scrolling and setting the colours should be inferred. The operation of outputing characters to the screen can be done by OPENing up a CON_ or SCR_ channel. This is part of the redirectable serial I/O system.

Screen output calls summary

D0	Name	Description
09	SD.EXTOP	Invokes additional routine as part of screen driver
0A	SD.PXENQ	Returns window size and cursor position (pixels)
0B	SD.CHENQ	Returns window size and cursor position (characters)
0C	SD.BORDR	Sets the border width and colour
0D	SD.WDEF	Redefines a window
0E	SD.CURE	Enables the cursor
0F	SD.CURS	Suppresses the cursor
10	SD.POS	Position cursor at row,column in character intervals
11	SD.TAB	Position cursor at any defined character column
12	SD.NL	Position cursor on a new line of characters

13	SD.PCOL	Position cursor on previous character column
14	SD.NCOL	Position cursor on next character column
15	SD.PROW	Position cursor on previous character row
16	SD.NROW	Position cursor on next character row
17	SD.PIXP	Position cursor using pixel coordinates
18	SD.SCROL	Scrolls all of a window
19	SD.SCRTP	Scrolls the top of a window
1A	SD.SCRBT	Scrolls the bottom of a window
1B	SD.PAN	Pans all of a window
1E	SD.PANLN	Pans cursor line
1F	SD.PANRT	Pans right hand end of cursor line
20	SD.CLEAR	Clears all of a window
21	SD.CLRTP	Clears the top of a window
22	SD.CLRBT	Clears the bottom of a window
23	SD.CLRLN	Clears the cursor line
24	SD.CLRRT	Clears right hand end of cursor line
25	SD.FOUNT	Sets or resets the fount
26	SD.RECOL	Recolours a window
27	SD.SETPA	Sets the paper colour
28	SD.SETST	Sets the strip colour
29	SD.SETIN	Sets the ink colour
2A	SD.SETFL	Sets flashing
2B	SD.SETUL	Sets underscoring
2C	SD.SETMD	Sets the character writing or plotting mode
2D	SD.SETSZ	Sets the character size and spacing
2E	SD.FILL	Fills a rectangular block within window
30	SD.POINT	Plots a point
31	SD.LINE	Plots a line
32	SD.ARC	Plots an arc
33	SD.ELLIPS	Plots an ellipse
34	SD.SCALE	Sets window scale
35	SD.FLOOD	Turns area flood on and off
36	SD.GCUR	Sets graphics cursor position

7.1.3 File handling

File handling can largely be carried out through the redirectable I/O system. For example, entire files can be transferred between microdrives and the RS232 ports without any problems. There are two basic areas of file handling which are dealt with in chapter 7. These are firstly loading and saving complete files, and secondly allowing direct access and record structured files to be dealt with.

File handling summary

D0	Name	escription
40	FS.CHECK	Checks all pending operations on a file
41	FS.FLUSH	Flushes buffers for a file
42	FS.POSAB	Positions file pointer at absolute location in file
43	FS.POSRE	Positions file pointer at relative location in file
45	FS.MDINF	Gets information about the storage medium
46	FS.HEADS	Sets the file header
47	FS.HEADR	Reads the file header
48	FS.LOAD	Loads a file into memory
49	FS.SAVE	Saves a file from memory

7.2 Serial I/O Calls

Serial I/O refers to the transfer of bytes or characters. This transfer is fully redirectable, so output can be switched from the screen to a printer (for example), simply be redefining the transfer channel. The detailed processing of data for any particular peripheral is carried out by a *device driver*. This piece of software ensures that QDOS can talk to any peripheral in a standard way, even though the actual operating details of these peripherals may vary widely.

Single byte I/O transfers would normally be used whenever the Job dealing with the data must control the action of the driver. In such cases, bytes which represent embedded control codes must be processed by the Job and not the driver.

The alternative to sending individual bytes is to send complete strings. Transferring files with strings is **much** more efficient than transferring files a byte at a time. When a string is read with a timeout of zero, only as many characters as are available at any particular time will be read. String transfers should be allowed to use as much of the internal memory for a buffer as is viable. If the buffer is too short, the transfer process between files will be inefficient (unless the file only contains a very small number of bytes).

It is important to use the timeout facility with care whenever more than one Job is accessing the same channel. This is because of the problem of a possible access clash. For example, if one Job is waiting for IO and another Job requests IO on the same channel, then, provided that the timeout is non-zero, the second Job will be rescheduled. The second Job will only be resumed when the first Job is no longer waiting (because it has completed the task or timed out). Note that the timeout period for the second Job commences from the time it *actually* gets access to the channel and not from the time when it first tried to get access.

7.2.1 Serial I/O return options

Serial I/O system calls can be defined to have three different types of return, depending on the application. The options are:

Wait for completion

This return option will not return until the required process is completed. This type of call should only be used with care, since, in the event of the process remaining incomplete, the calling Job may remain inactive for good.

Return immediately

This is the opposite of the wait for completion option. When the call is made, all processes which can be effected immediately will be performed. Anything which cannot be done instantly will remain unfinished. Note that if an output call returns incomplete (ie. the channel output buffer is temporarily full), it will remain incomplete. The original data will not have been sent, and must be resent at a later time if it is not to be lost.

Wait until timeout or completion

This option will return either after a predefined timeout period, or when the task is complete (if this should occur before the end of the timeout period). An example of the use of such a timeout might be that of an RS232 link to a keyboard. If no key is pressed on the keyboard within say 10 minutes, the read from keyboard call will timeout, and could disconnect the keyboard from the QL. Note that all times are calculated in terms of the 50 or 60 Hertz display refresh period. The timeout period for all serial I/O calls is passed in D3 and is not modified by the traps.

General

All of the above return options are available through one call. Each is characterised only by the maximum permissible length of wait. The waiting period can be zero, defined or infinite. A word is used for the defined number of frame cycles before return, which can therefore take a maximum value of 32767 (or about eleven minutes). A timeout period of −1 indicates that an infinte wait should be provided. Other negative values should not be used.

IO.PEND TRAP#3 D0=0

Check for pending input

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
EF	end of file

Description:

This trap checks for pending input on a channel. Information is returned to the calling routine by the errors only. It does not actually read any data or modify the input channel in any way.

IO.FBYTE TRAP#3 D0=1

Fetch a byte

Call parameters		Return parameters	
D1		D1.B	byte fetched
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
EF	end of file

Description:

This trap will fetch a byte from a defined channel with the programmed timeout. The byte which has been fetched (if there is one) will be returned in register D1.

IO.FLINE TRAP#3 D0=2

Fetch line of characters terminated by <LF>

Call parameters		Return parameters	
D1		D1.W	number of bytes fetched
D2.W	length of buffer	D2.W	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of buffer	A1	updated pointer to buffer
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
EF	end of file
BO	buffer overflow

Description:

A line of characters can be fetched by this trap. The number of characters in the line (including the <LF> if found) are returned in D1. For all multiple byte reads, the timeout period is the maximum time allowed from the issuing of the trap. It is not the time between consecutive bytes being issued. Timeout cannot therefore be used to detect, for example, if the operator is typing slowly. The maximum number of bytes which can be fetched within the time limit are fetched.

Fetching a line of characters from the console IO device has a special significance. All characters are echoed on the associated screen window as they are typed at the keyboard. Whenever the fetch a line trap is made, the cursor is enabled in the window, and it is disabled when the line has been read. The cursor keys can be used to modify the line in the normal way, so:

<	moves the cursor left
>	moves the cursor right
control <	deletes the character to the left
control >	deletes the character to the right

IO.FSTRG TRAP#3 D0=3

Fetch a string of bytes

Call parameters		Return parameters	
D1		D1.W	number of bytes fetched
D2.W	length of buffer	D2.W	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of buffer	A1	updated pointer to buffer
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
EF	end of file

Description:

This trap will fetch a string of bytes from the allocated channel. If this happens to be the keyboard, characters will not be echoed in any window and any cursor handling is left up to the application program. Window echo and cursor handling can be implemented by using IO.FLINE (trap#3, D0=2). If a pointer to an array of bytes is passed in A1, then A1 will point to the next byte on return.

IO.EDLIN TRAP#3 D0=4

Edit a line of characters (console driver only)

Call parameters		Return parameters	
D1.L	cursor/line length	D1.L	cursor/line length
D2.W	length of buffer	D2	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	pointer to end of line	A1	pointer to end of line
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
BO	buffer overflow

Description:

This trap is very similar to IO.FLINE, but with some notable differences. It can be used to edit a line. A1 should always be set to point to the end of the line. D1 is set to contain the current cursor position in the top word, and the current length of the line in the bottom word. So that the line can be edited easily, it is written out (from the current cursor position) to the console every time that this trap is made. When the trap is made, the line should not have a terminating character. However, upon return, the terminating character will be included in the character count. All of <CTRL J>, <ENTER>, UP cursor or DOWN cursor can be used as valid terminating characters.

IO.SBYTE TRAP#3 D0=5

Send a byte

Call parameters		Return parameters	
D1.B	byte to be sent	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
DF	drive full
OR	out of range (off window, paper etc.)

Description:

Single bytes can be sent to a channel with this trap.

For certain devices, the wait on output may not operate in the expected manner. Unexpected behaviour occurs whenever the output device includes any form of buffering (eg. the buffer on a printer). Internal buffering on an output device is totally invisible to the I/O system. A return which indicates that the I/O transaction has been completed just means that the bytes have been transferred to the device driver. This does not necessarily imply that they have come out at the other end of the device (eg. been printed).

See also IO.SSTRG

IO.SSTRG TRAP#3 D0=7

Send a string of bytes

Call parameters		Return parameters	
D1		D1.W	number of bytes sent
D2.W	number of bytes to send	D2.W	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of buffer	A1	updated pointer to buffer
A2		A2	preserved
A3		A3	preserved

Error returns:

NC	not complete
NO	channel not open
DF	drive full

Description:

Strings of bytes can be sent to a channel with this trap.

Several special provisions have been made for generating newlines during output to the console or screen devices. Automatic insertion of newlines will occur whenever an <LF> is sent, or if the cursor reaches the right hand side of the screen. Unlike the send byte trap, this send string trap doesn't generate the OR error. In the event of the cursor having been suppressed, the newline is not effected immediately, but is held pending. It can be activated by:

sending another byte or string,
changing the character size,
activating the cursor,
or requesting the position of the cursor.

Pending newlines are cancelled by any call to position the cursor.

See also IO.SBYTE

7.3 Screen Output Control

Screen output control covers the operations of modifying the window, controlling the cursor, clearing part or all of the window, scrolling, and setting the colours. Since these operations are all specific to the screen, these traps are not part of the basic redirectable I/O system. Having said this, it is nevertheless possible to write screen emulating drivers which can be accessed via trap#3.

7.3.1 User screen drivers

SD.EXTOP TRAP#3 D0=9

Calls an extended operation

Call parameters		Return parameters	
D1	supplied to user routine	D1	returned from user routine
D2	supplied to user routine	D2	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	supplied to user routine	A1	returned from user routine
A2	routine start address	A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

User supplied routines can be called using this command as if they were part of the standard screen driver. Timeout and channel ID are passed in D3 and A0 as usual. A2 is set to contain the start address for the user supplied routine. The other registers D1, D2 and A1 can all be allocated by the applications programs as required.

7.3.2 Finding the current window size

SD.PXENQ TRAP#3 D0=A

Get window size and cursor position (in pixels)

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of enquiry block	A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

The size of the window attached to the defined channel can be found with this trap. The current cursor position within that window is also returned. All positions are returned in pixel co-ordinates. Size and position are returned in a four word enquiry block. A1 is set to point to the base of this block, so the format is:

base	X dimension of window
base+2	Y dimension of window
base+4	X position of cursor
base+6	Y position of cursor

Note that the top left cursor position is 0,0.

SD.CHENQ TRAP#3 D0=B

Get window size and cursor position (in characters)

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of enquiry block	A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open

Description:

The size of the window attached to the defined channel can be found with this trap. The current cursor position within that window is also returned. All positions are returned in character co-ordinates. Size and position are returned in a four word enquiry block. A1 is set to point to the base of this block, so the format is:

base	X dimension of window
base+2	Y dimension of window
base+4	X position of cursor
base+6	Y position of cursor

Note that the top left cursor position is 0,0.

7.3.3 Window control

SD.BORDR TRAP#3 D0=C

Sets the border width and colour

Call parameters		Return parameters	
D1.B	colour	D1	undefined
D2.W	width	D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	preserved
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This call is used to redefine the border of a window. The default border width is 0. Border width is doubled on the vertical edges, and falls **inside** the original window limits. Changing the border width causes the cursor to be homed.

All subsequent traps (except this one) to define the cursor position and window limits use the new reduced window size.

The colour definitions are tabulated in section 7.3.9. Note that a new special colour definition is also available on this particular trap. Defining the colour as $80 will produce a *transparent* border, which means that the original contents of the border are not altered.

SD.WDEF TRAP#3 D0=D

Redefine a window

Call parameters		Return parameters	
D1.B	border colour	D1	undefined
D2.W	border width	D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of window block	A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open
OR	window will not fit on page

Description:

This call allows the position and shape of a window to be redefined. The old window contents are not moved or modified, but the cursor is repositioned at the top left hand corner of the new window. The window block should contain 4 words as follows:

base	window width
base+2	window height
base+4	X origin
base+6	Y origin

SD.CURE TRAP#3 D0=E

Enable the cursor

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap will cause the cursor to be enabled on the window defined by the channel ID. Whenever a read line or edit line trap is issued to a window, the cursor will be enabled automatically. Note that an error will not be returned if the cursor was already enabled. This call merely ensures that the cursor is enabled.

SD.CURS TRAP#3 D0=F

Suppress the cursor

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	preserved
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This call will cause the cursor to be turned off on the selected channel. If the cursor was already turned off, then this call will not return an error.

SD.POS TRAP#3 D0=10

Position cursor at row, column (character co-ordinates)

Call parameters		Return parameters	
D1.W	column number	D1	undefined
D2.W	row number	D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

This trap positions the cursor at an absolute position on the screen using character co-ordinates. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.TAB TRAP#3 D0=11

Positions the cursor at a character column

Call parameters		Return parameters	
D1.W	column number	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

This trap will TAB the cursor to the column number passed in D1. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.NL TRAP#3 D0=12

Positions cursor on a new line of characters

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

This trap will force a newline and thereby position the cursor at the start of the next row of characters. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.PCOL TRAP#3 D0=13

Position cursor on previous character column

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

Moves the cursor one character cell to the left. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.NCOL TRAP#3 D0=14

Position cursor on next character column

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

Moves the cursor one character cell to the right. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.PROW TRAP#3 D0=15

Position cursor on the previous character row

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

Moves the cursor onto the line above. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.NROW TRAP#3 D0=16

Position cursor on the next character row

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	

Error returns:

NC	not complete
NO	channel not open
OR	position would be outside of the window

Description:

Moves the cursor onto the line below. The cursor position is the top lefthand corner of the **next** character rectangle. Character rectangles are all referenced relative to the top lefthand corner of the window. If the trap returns an error, the cursor position will not be changed. This trap causes any pending newlines in the window to be cancelled.

SD.PIXP TRAP#3 D0=17

Position cursor using pixel co-ordinates

Call parameters		Return parameters	
D1.W	X co-ordinate	D1	undefined
D2.W	Y co-ordinate	D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open
OR	off the window

Description:

The cursor can be positioned within a window using this trap. All co-ordinates are relative to the top lefthand corner of the window, which is at 0,0. A pending newline within the window will be cleared. If an error is returned, the cursor isn't moved.

7.3.5 Scrolling windows

SD.SCROL TRAP#3 D0=18

Scrolls all of a window

Call parameters		Return parameters	
D1.W	distance to scroll	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap causes the entire contents of the window to be scrolled. Scrolling consists of transferring pixels from one row to another. Rows which have been vacated during the scrolling process are filled with the paper colour. A positive scrolling distance implies that the pixels in the window will move downwards. A text window would normally be scrolled in the upward direction, and would require a negative distance argument.

See also SD.SCRTP and SD.SCRBT for partial scrolling.

SD.SCRTP TRAP#3 D0=19

Scrolls the top part of a window

Call parameters		Return parameters	
D1.W	distance to scroll	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap causes the contents of the top half of the window to be scrolled. The division between the top and the bottom of the screen is the cursor line, which is not scrolled. The cursor is not moved. Scrolling consists of transferring pixels from one row to another. Rows which have been vacated during the scrolling process are filled with the paper colour. A positive scrolling distance implies that the pixels in the window will move downwards. A text window would normally be scrolled in the upward direction, and would require a negative distance argument.

SD.SCRBT TRAP#3 D0=1A

Scrolls the bottom part of a window

Call parameters		Return parameters	
D1.W	distance to scroll	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap causes the contents of the bottom half of the window to be scrolled. The bottom part of the screen is all that part below the cursor line. The cursor is not moved. Scrolling consists of transferring pixels from one row to another. Rows which have been vacated during the scrolling process are filled with the paper colour. A positive scrolling distance implies that the pixels in the window will move downwards. A text window would normally be scrolled in the upward direction, and would require a negative distance argument.

7.3.6 Panning windows

SD.PAN TRAP#3 D0=1B

Pans all of a window sideways

Call parameters		Return parameters	
D1.W	distance to pan	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap causes the whole of a window to be panned by the desired number of pixels to the left (with a negative argument) or right (with a positive argument). The space left behind is filled with the paper colour.

SD.PANLN TRAP#3 D0=1E

Pans the cursor line

Call parameters		Return parameters	
D1.W	distance to pan	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap causes the whole of the cursor line to be panned by the desired number of pixels to the left (with a negative argument) or right (with a positive argument). The space left behind is filled with the paper colour. The cursor line will be either 10 or 20 rows deep depending upon the particular character size which is selected.

SD.PANRT TRAP#3 D0=1F

Pans righthand end of cursor line

Call parameters		Return parameters	
D1.W	distance to pan	D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

This trap causes the righthand end of the cursor line to be panned by the desired number of pixels to the left (negative argument) or right (positive argument). The space left behind is filled with the paper colour. The righthand end includes the character which is at the current cursor position.

7.3.7 Clearing windows

SD.CLEAR TRAP#3 D0=20

Clears all of a window

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

The entire window can be cleared using this trap. All pixels in the window are overwritten with the paper colour. Other parts of the window can also be selectively cleared. Traps SD.CLRTP, SD.CLRBT, SD.CLRLN and SD.CLRRT will clear the top, bottom, cursor line and righthand end of cursor line respectively. Note that the division between the top and bottom of the window is the cursor line. The cursor line itself is not considered to be inside either the top or the bottom part of the screen.

SD.CLRTP TRAP#3 D0=21

Clears the top of a window

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

The top of a window can be cleared using this trap. All pixels in the designated area are overwritten with the paper colour. The cursor line provides the division between the top and the bottom of the window (but is itself included in neither). Other parts of the window can also be selectively cleared.

SD.CLRBT TRAP#3 D0=22

Clears the bottom of a window

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

The bottom of a window can be cleared using this trap. All pixels in the designated area are overwritten with the paper colour. The cursor line provides the division between the top and the bottom of the window (but is itself included in neither). Other parts of the window can also be selectively cleared.

SD.CLRLN TRAP#3 D0=23

Clears the cursor line

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

The cursor line can be cleared using this call. The line will be overwritten with pixels in the paper colour. The whole height of the current character size (10 or 20 pixels) is cleared by this trap.

SD.CLRRT TRAP#3 D0=24

Clears the righthand end of the cursor line

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:

The righthand end of the cursor line can be cleared using this call. The righthand side of the line will be overwritten with pixels in the paper colour (including the character at the current cursor position). The whole height of the current character size (10 or 20 pixels) is cleared by this trap.

7.3.8 Setting the character font

SD.FOUNT TRAP#3 D0=25

Sets or resets the character font

Call parameters		Return parameters	
D1		D1	undefined
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
A0	channel ID	A0	preserved
A1	base of font	A1	undefined
A2	base of second font	A2	preserved

Error returns:

NC	not complete
NO	channel not open

Description:
Characters which are to be displayed on the screen are stored as a 5x9 array of pixels in a 6x10 rectangle. This ensures that no two characters will ever touch when placed in adjacent character cells. There is a default font and a second font built into the QL, but alternative user defined fonts can also be selected. The default font extends from ASCII character $20 to $7F, and the default second font extends from $80 to $BF.
When changing the font, the default font will be selected if the base address is given as 0. The font must be stored in memory in a special format. Below a certain value, characters will be invalid (default $1E). From the next value (default $1F) a known number of characters are valid (default $61). The format is therefore as follows:

$00	lowest valid character (byte)
$01	number of valid characters−1 (byte)
$02 to $0A	9 bytes of pixels for 1st valid character
$0B to $13	9 bytes of pixels for 2nd valid character
$13 to $1B	etc.