6 INPUT/OUTPUT OPERATIONS (TRAP #3)

Input and output QDOS procedures fall into two major categories. First, there are those that are concerned with the actual allocation of channels for devices and files. Second, there are those that actually perform input and output operations on the allocated channels. The first category comprises the TRAP #2 calls, as discussed in Chapter 5. The second category comprises the TRAP #3 calls which we will discuss now.

6.1 Timeouts

All TRAP #3 procedures require a timeout to be specified. The timeout period is a multiple of the monitor frame rate (i.e., 50/60 Hz). A timeout period of unity is therefore equivalent to 20 ms for a 50 Hz timebase, and 16.666 ms for a 60 Hz timebase.

If a routine is called with the timeout set to zero, the call will return immediately after attempting its task, regardless of whether or not it succeeded. A positive timeout will make the call return immediately on completion, or at the end of the timeout, whichever occurs first. A maximum timeout period of 32767 times the unit timebase period is permitted. This gives a maximum period of 10 minutes, 55.3 seconds for a 50 Hz timebase, and 9 minutes, 6.1 seconds for a 60Hz timebase. A timeout value of -1 signifies that an indefinite period is required. No other negative value should be used.

The IO.FBYTE procedure is a worthy example. Suppose the procedure was being used to input a byte (character) from the keyboard. If a timeout of zero is used, the procedure will return a character if one is waiting, or return the error ERR.NC (not complete) if no character exists. Either way, an immediate return is made. A positive timeout will force the routine to look for a character for the specified length of time. If a character is entered within the time, an immediate return is made with the character. If no character arrives within the time specified, the routine will return at the end of the timeout with the ERR.NC error. A timeout of -1 will force the procedure to wait until complete, that is, wait until a character is entered.

6.2 Important principles

It must be remembered that QDOS supports multi-tasking and, therefore, there may be more than one job trying to get access to a particular

channel. If your job call requests access to a channel that is already being waited on, one of two things will happen. If your timeout was given as zero, your call will return immediately as incomplete. If your timeout was not zero, your job will get re-scheduled until such time as the I/O channel request can be serviced. Note particularly that your specified timeout period commences from the moment your job obtains access to the channel, and not from the moment the original request was made. This means, of course, that the actual timeout period may be longer than the programmed timeout.

Incompletion of a job has important side-effects when dealing with output. In such cases, incompletion means that the QDOS procedure could not finish outputting its data. Any data not sent, because of an incomplete return from a procedure, must be re-sent at some later stage. The case of single character (byte) output is trivial; you would simply re-send the byte. When a string of characters is being sent, the QDOS procedure will return a count of the number of characters actually sent.

The input and output of large amounts of data should be handled carefully. It is inefficient to perform such I/O a byte at a time. The QL comes with a large amount of RAM and reasonable decisions over the use of memory buffers should be made. Characters would then be imported and exported in as large a block as possible at any one time.

6.3 Screen channel definition blocks

Approximately 70% of QDOS TRAP #3 procedures are related to screen operations. The procedure to redefine a window (SD.WDEF) is one example. All of the procedures require a channel ID to be specified. The topic of channel IDs and screen channels is, therefore, so important that we will look at these in some detail.

THE CHANNEL ID

A channel ID is a long-word holding two bits of information (no pun intended!). First, in the high order word of the ID, there is a tag value. This value will get incremented each time a channel is opened. Second, in the low order word of the ID, there is a channel index code. This code is used internally by QDOS. For information only (you would never normally use the fact), the channel value, stored in the ID, multiplied by four will supply an appropriate index into a channel table (see Fig.6.1).

When the machine is first switched on, there are three screen channels open. In SuperBASIC the channels are designated #0, #1, and #2. The actual correspondence between these SuperBASIC screen channels and the QDOS channel IDs will be as follows:

SuperBASIC #	QDOS ID		
	HEX.		DENARY
	TAG	CHAN.	
0	0000	0000	0
1	0001	0001	65537
2	0002	0002	131074

If, for example, SuperBASIC channel #2 is re-opened, it will still be channel #2 as far as SuperBASIC is concerned, but its internal QDOS ID will be $00030002 (denary 196610).

It is important to realise that, in practice, you should never need to know this correspondence. When writing pure assembly language application programs you will always have a copy of the QDOS channel ID, and that is all you need. If you are passing a SuperBASIC channel number over to an assembly language utility, the utility will collect the standard channel integer (i.e., #0, #1, etc.) and use it to calculate the internal QDOS channel ID (see Chapters 8 and 11).

SCREEN CHANNELS

QDOS maintains a channel resource management table, which holds such information as the highest known channel number, and so on (see Fig.6.1). Two pointers within this table, SV_CHBAS and SV_CHTOP, point to the base and top of the channel table respectively. The pointers within the channel table (long-words) point, in turn, to the corresponding channel definition blocks. Figure 6.1 shows the layout of screen channel definition blocks.

If it is a requirement of an applications program to obtain information about a particular screen channel, it should be obtained by calling your 'get the information' subroutine via the TRAP #3 extended operation procedure (SD.EXTOP; D0=9). The SD.EXTOP call will assume that your subroutine is a standard device driver and perform a couple of important operations. First, it will pass the base of the system variables (SV_BASE) to the subroutine in register A6. Second, register A7 will be set to the supervisor stack in such a way that you may use up to 64 bytes of stack space within your subroutine. Note that your subroutine actually will be running in supervisor mode. Third, the channel ID passed over to SD.EXTOP in register A0 will be converted to a pointer (in A0) to the base of the corresponding channel definition block. These operations provide your subroutine with vital pointer information in a very neat and convenient manner. When the return from subroutine (RTS) instruction is executed at the end of your subroutine (note: RTS not RTE), register A0 will be reset to the channel ID specified on entry to SD.EXTOP.

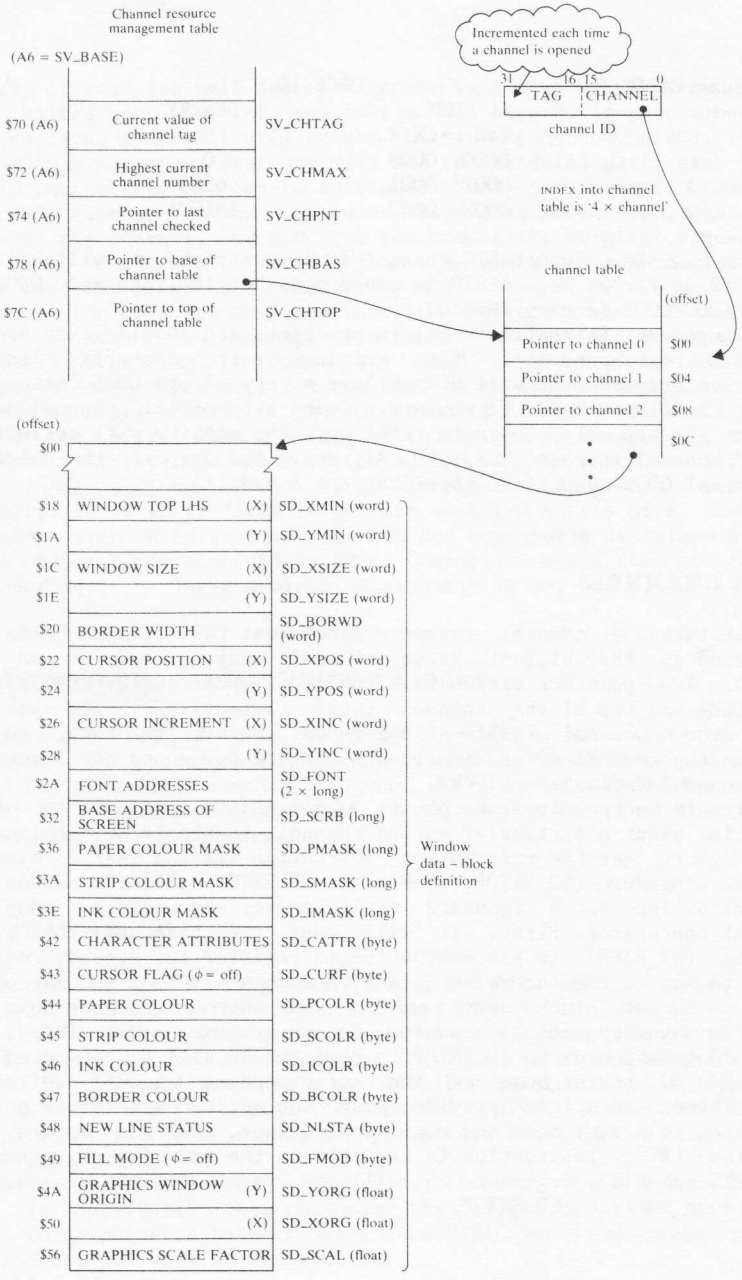

Figure 6.1 Screen channels and definition block

6.4 Colour

A number of the TRAP #3 procedures allow a colour to be specified as one of their parameters. Setting the ink colour (SD.SETIN) is one example of this. Three colours are used by the screen drivers. There is the 'paper' colour which is the background colour; the 'ink' colour which is the main printing and graphical plotting colour; and the 'strip' colour used for highlighting.

The solid colours which may be specified for the two screen modes are as follows:

Mode 8 (256)		Mode 4 (512)	
0	- black	0	- black
1	- blue	1	- black
2	- red	2	- red
3	- magenta	3	- red
4	- green	4	- green
5	- cyan	5	- green
6	- yellow	6	- white
7	- white	7	- white

In addition to the solid colours there are four stipple patterns that can be specified. Stipple patterns exist within a 2x2 pixel matrix, and can be selected by setting appropriate bits in the colour byte (see Fig.6.2).

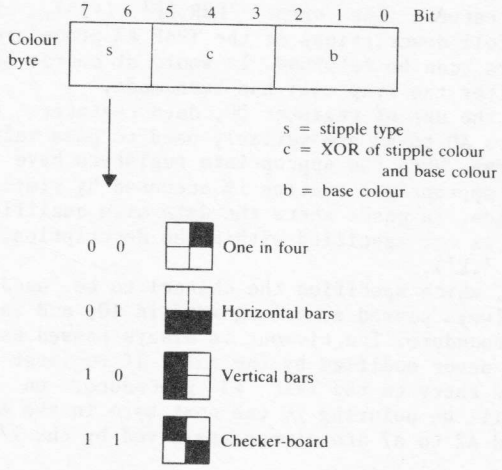

Figure 6.2 Exploded view of colour byte

Bits 7 and 6 will specify the stipple pattern as shown. Bits 0 to 2 specify the base colour (0..7), and bits 3 to 5 hold a value corresponding to the XOR (exclusive_OR) of the base colour and the stipple pattern colour. Suppose we want a base colour of blue and a stipple colour of yellow. The base colour has the binary value '001', and the stipple colour the binary value '110'. The XOR of these two

values is the binary value '111', and it is this value that must be put into bits 3 to 5 of the colour byte. If we also wanted a checker-board type of stipple, the whole colour byte would be binary encoded as '11 111 001', which is the denary colour 249.

This method of encoding the pattern colours enables the standard solid colours to be obtained quite naturally. For example, take the solid colour green (value 4). This has a colour byte binary pattern of '00 000 100'. The stipple pattern is type 0 (one dot in four). The XOR value is zero and, therefore, the stipple colour must be 4 (binary 100), which is green! This gives us a base colour and a stipple colour that are the same, and hence we get our solid colour.

6.5 Use of 68000 registers

The TRAP #3 procedures are accessed with register D0 (byte) indicating which particular call is required. This register is used also to return an error status (long-word) to the calling process. If the error code returned is not zero then an error has occurred. Small negative error codes are used to indicate standard errors. These error codes are listed in Appendix C. If the trap call invoked some form of additional device driver, the error code returned can be a pointer to a specific error message. In order that the two types of error return code might never be confused, the pointer type error code is in fact a pointer to an address $8000 below that of the true error message. Potentially, all QDOS routines can return the error 'ERR.BP' (-15), signifying 'bad parameter'. The full descriptions of the TRAP #3 procedures state which additional errors can be returned. It would of course be wise to check for any errors after the trap call has been made.

In addition to the use of register D0, data registers D1 to D3 and address registers A0 to A3 are variably used to pass values to and from the QDOS procedures. When the appropriate registers have been set for any one call the appropriate routine is accessed by simply executing the TRAP #3 instruction. In cases where the data size qualifier (i.e., '.B', '.W', or '.L') is not specified within the description, the default is long-word (i.e., '.L').

The channel ID, which specifies the channel to be used for the I/O transfer, is always passed as a long-word in A0, and is never modified by the TRAP #3 procedure. The timeout is always passed as a word in D3, and it also is never modified by the call. If register A1 points to an array of bytes on entry to the TRAP #3 procedure, on exit from the procedure A1 will be pointing to the next byte in the array. Registers D2 to D7, A0, and A2 to A7 are always preserved by the I/O procedures.

IO.PEND $00 (0)

Check for pending input

Entry parameters: D3.W Timeout
A0 Channel ID

Return parameters: none

Affected registers: D1, A1

Additional errors: NC (-1) not complete (no pending input)
NO (-6) channel not open
EF (-10) end of file

Description

This trap call can be used to check for pending input on a channel. Note that only a check for input is performed; the input channel is not modified in any way, and no input of data is performed.

IO.FBYTE $01 (1)

Fetch a byte

Entry parameters:	D3.W	Timeout
	A0	Channel ID
Return parameters:	D1.B	Byte fetched
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open
	EF (-10)	end of file

Description

This procedure will fetch a byte from the specified input channel.

IO.FLINE $02 (2)

Fetch a line (terminated by ASCII <LF>)

Entry parameters:	D2.W	Length of buffer
	D3.W	Timeout
	A0	Channel ID
	A1	Base of buffer
Return parameters:	D1.W	Number of bytes fetched
	A1	Updated buffer pointer
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	BO (-5)	buffer overflow
	NO (-6)	channel not open
	EF (-10)	end of file

Description

When dealing with console input, this procedure has special properties. First, the characters read from the keyboard will be echoed in the appropriate window. Second, the standard cursor keys (LEFT and RIGHT) can be used for simple editing as follows:

LEFT	-	**move cursor left**
RIGHT	-	**move cursor right**
CTRL-LEFT	-	**delete character left**
CTRL-RIGHT	-	**delete character under cursor**

Note that the cursor, within the specified window, will only be enabled for the duration of the procedure call. The count of the number of bytes fetched, as returned in D1, will include the line terminator (ASCII <LF>) if it was found.

The line-feed terminator may not be found if the timeout is exhausted, and will never be found if the buffer overflows. In such cases the cursor will be left enabled.

On exit, the pointer in register A1 will point to the byte following the last character entered.

IO.FSTRG $03 (3)

Fetch a string of bytes

Entry parameters:	D2.W	Length of buffer
	D3.W	Timeout
	A0	Channel ID
	A1	Base of buffer
Return parameters:	D1.W	Number of bytes fetched
	A1	Updated buffer pointer
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open
	EF (-10)	end of file

Description

This procedure will fetch a string or block of bytes from the specified input channel. The bytes fetched will not be echoed on the screen, even if the channel device is a window. A return will be effected either when the timeout is exhausted or when the buffer becomes full.

IO.EDLIN $04 (4)

Edit a line (console only)

Entry parameters:	D1.L	Line/cursor parameters
	D2.W	Length of buffer
	D3.W	Timeout
	A0	Channel ID
	A1	Pointer to end of line
Return parameters:	D1	Line/cursor parameters
	A1	Pointer to end of line
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	BO (-5)	buffer overflow
	NO (-6)	channel not open

Description

This is similar to the procedure IO.FLINE ($02) except that an initial line, on which to start the editing, is supplied to the user. On entry, register D1 must hold two words of information about the line. The high order word must contain the current cursor position (0..n), and the low order word must specify the total line length. The line specified should not contain the terminating character (ASCII <LF>). Only the part of the line starting from the current cursor position up to the end of the line will be given to the user.

On exit, the procedure will have updated the parameters in registers D1 and A1 to correspond to the edited line. Valid terminating characters are <LF>, <CURSOR-UP>, and <CURSOR-DOWN>. The terminating character will be included in the line (and hence the line length) when the procedure returns. Note that the pointer in register A1 always points to the character byte following the last character entered.

IO.SBYTE $05 (5)

Send a byte

Entry parameters:	D1.B	Byte to be sent
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	out of range
	NO (-6)	channel not open
	DF (-11)	drive full

Description

This procedure will send a byte out to the specified output channel.
Special provisions exist for the output of a line-feed (<LF>) to a screen or console device. First, a newline is inserted on receipt of the line-feed terminator, or when the cursor reaches the right-hand side of the window. If the cursor is suppressed, the newline will be held pending. To release it one of the following may be performed:

1. another byte may be sent
2. the character size may be changed
3. the cursor may be enabled
4. the cursor position may be requested

If the cursor is explicitly positioned, the pending newline will be cancelled. Note especially that an explicit newline will replace an implicit one, thus providing sensible output with no unwanted blank lines.

The error code ERR.OR (-4) will be returned if any newline operation has taken place.

IO.SSTRG $07 (7)

Send a string of bytes

Entry parameters:	D2.W	Number of bytes to send
	D3.W	Timeout
	A0	Channel ID
	A1	Base of buffer
Return parameters:	D1.W	Number of bytes sent
	A1	Updated pointer to buffer
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open
	DF (-11)	drive full

Description

This procedure will send a string of bytes out to the specified channel. Special provisions exist for the output of a line-feed (<LF>) to a screen or console device. First, a newline is inserted on receipt of the line-feed terminator, or when the cursor reaches the right-hand side of the window. If the cursor is suppressed, the newline will be held pending. To release it one of the following may be performed:

1. another string may be sent
2. the character size may be changed
3. the cursor may be enabled
4. the cursor position may be requested

If the cursor is explicitly positioned, the pending newline will be cancelled. Note especially that an explicit newline will replace an implicit one, thus providing sensible output with no unwanted blank lines.

SD.EXTOP $09 (9)

Call an extended operation

Entry parameters:	D1	Parameter (if required)
	D2	Parameter (if required)
	D3.W	Timeout
	A0	Channel ID
	A1	Parameter (if required)
	A2	Address of routine
Return parameters:	D1	Parameter (if used)
	A1	Parameter (if used)
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This trap call provides a mechanism for accessing a user supplied routine in supervisor mode. The routine specified on entry must conform to device driver rules. A detailed description of device driver routines is outside the scope of this book, but Sec.6.3 discusses the pertinent points.

The registers available for parameter passing (i.e., the three for importing, and the two for exporting) do not have to be used. It is simply the case that the values in these registers will not become corrupted in the 'interface' between the TRAP routine and the user-supplied device driver.

SD.PXENQ $0A (10)

Return window size & cursor position in pixel coords.

Entry parameters:	D3.W	Timeout
	A0	Channel ID
	A1	Base of enquiry block
Return parameters:	(Updated enquiry block)	
Affected registers:	A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

An enquiry block is required for this procedure call, consisting of four words:

Block offset	Use
$00	**Window size (X)**
$02	**Window size (Y)**
$04	**Cursor position (X)**
$06	**Cursor position (Y)**

The top left-hand corner of the window will have the coordinates [0,0]. If a newline is pending on the specified window channel, it will be activated (i.e., released).

SD.CHENQ $0B (11)

Return window size & cursor position in character coords.

Entry parameters:

D3.W	Timeout
A0	Channel ID
A1	Base of enquiry block

Return parameters: (Updated enquiry block)

Affected registers: A1

Additional errors:

NC (-1)	not complete
NO (-6)	channel not open

Description

An enquiry block is required for this procedure call, consisting of four words:

Block offset	Use
$00	**Window size (X)**
$02	**Window size (Y)**
$04	**Cursor position (X)**
$06	**Cursor position (Y)**

The top left-hand corner of the window will have the coordinates [0,0]. If a newline is pending on the specified window channel, it will be activated (i.e., released).

SD.BORDR $0C (12)

Set border width & colour

Entry parameters:	D1.B	Colour
	D2.W	Width
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This procedure will redefine the border of the specified window. A border is, by default, of zero width. When the border is set up it will lie inside the window limits, and the vertical edges will be of double width. All subsequent screen calls will use the reduced window size (except a subsequent call to SD.BORDR). The cursor will be homed if the border width is changed.

The standard colour codes are as described in Sec.6.4. There is also the colour $80 (128) which is treated as a special case, and will create a transparent border, leaving the original border contents intact.

SD.WDEF $0D (13)

Define a window and its border

Entry parameters:	D1.B	Border colour
	D2.W	Border width
	D3.W	Timeout
	A0	Channel ID
	A1	Base of window block
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - window too big
	NO (-6)	channel not open

Description

This call is used to redefine the shape and position of a window. The original contents of the screen will not be changed or moved, but the cursor will be set to the top left-hand corner of the new window.

A window definition block consisting of four words must be set up before the TRAP call is made, and should contain:

Block offset	Use
$00	Window size (X)
$02	Window size (Y)
$04	Window origin (X)
$06	Window origin (Y)

The window origin corresponds to the top left-hand corner of the defined window.

SD.CURE $0E (14)

Enable the cursor

Entry parameters:	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This procedure will enable the cursor in the specified window channel. Note that the cursor will automatically be enabled when a 'read line' (IO.FLINE) or 'edit line' (IO.EDLIN) procedure is invoked.

SD.CURS $0F (15)

Suppress the cursor

Entry parameters:

D3.W	Timeout
A0	Channel ID

Return parameters: none

Affected registers: D1, A1

Additional errors:

NC (-1)	not complete
NO (-6)	channel not open

Description

This procedure will disable the cursor in the specified window channel. Note that the cursor will automatically be disabled when a 'read line' (IO.FLINE) or 'edit line' (IO.EDLIN) procedure terminates normally.

SD.POS $10 (16)

Move cursor absolute using character coordinates

Entry parameters:	D1.W	Column position
	D2.W	Row position
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will position the cursor at a specified absolute position. The top left-hand corner of the window is position [0,0].

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.TAB $11 (17)

Tabulate

Entry parameters:	D1.W	Column position
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will position the cursor at the specified tab-stop position. The specified position may be anywhere on the current cursor line.

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.NL $12 (18)

Newline

Entry parameters:	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will force a newline to be given in the specified window channel.

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.PCOL $13 (19)

Cursor back

Entry parameters:	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will backspace the cursor non-destructively (i.e., the cursor will not rub out the previous character).

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.NCOL $14 (20)

Cursor forward

Entry parameters:

D3.W	Timeout
A0	Channel ID

Return parameters: none

Affected registers: D1, A1

Additional errors:

NC (-1)	not complete
OR (-4)	range error - not in window
NO (-6)	channel not open

Description

This procedure will move the cursor forward one character position, non-destructively.

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.PROW $15 (21)

Cursor up

Entry parameters:	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will move the cursor up one line non-destructively. The column position of the cursor will be unchanged.

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.NROW $16 (22)

Cursor down

Entry parameters:	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will move the cursor down one line non-destructively. The column position of the cursor will be unchanged.

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.PIXP $17 (23)

Move cursor absolute pixel using pixel coordinates

Entry parameters:	D1.W	X coordinate
	D2.W	Y coordinate
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	OR (-4)	range error - not in window
	NO (-6)	channel not open

Description

This procedure will position the cursor at a specified absolute position. The top left-hand corner of the window is position [0,0]. Pixel coordinates should correspond to the top left-hand corner of the required character rectangle.

If a newline is pending, it will be cleared by this call. The original cursor position will not be altered if an error occurs.

SD.SCROL $18 (24)

Scroll entire window

Entry parameters:	D1.W	Distance to scroll
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This procedure will scroll the whole of the specified channel window. An upward scroll can be obtained by specifying a negative distance. The distance to scroll is always specified in terms of pixels. Vacated pixel rows will be filled with the 'paper' colour.

The cursor position will not be altered.

SD.SCRTP $19 (25)

Scroll top of window

Entry parameters:	D1.W	Distance to scroll
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This procedure will scroll the top part of the specified channel window. An upward scroll can be obtained by specifying a negative distance. The distance to scroll is always specified in terms of pixels. Vacated pixel rows will be filled with the 'paper' colour.

The top part of the window is defined as the area of the window above (and not including) the cursor line. The cursor position will not be altered.

SD.SCRBT $1A (26)

Scroll bottom of window

Entry parameters:	D1.W	Distance to scroll
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This procedure will scroll the bottom part of the specified channel window. An upward scroll can be obtained by specifying a negative distance. The distance to scroll is always specified in terms of pixels. Vacated pixel rows will be filled with the 'paper' colour.

The bottom part of the window is defined as the area of the window below (and not including) the cursor line. The cursor position will not be altered.

SD.PAN $1B (27)

Pan entire window

Entry parameters:	D1.W	Distance to pan
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1)	not complete
	NO (-6)	channel not open

Description

This procedure will pan the whole of the specified channel window. A pan to the left can be obtained by specifying a negative distance. The distance to pan is always specified in terms of pixels. Vacated pixel positions will be filled with the 'paper' colour.

The cursor position will not be altered.

SD.PANLN $1E (30)

Pan cursor line

Entry parameters:

D1.W	Distance to pan
D3.W	Timeout
A0	Channel ID

Return parameters: none

Affected registers: D1, A1

Additional errors:

NC (-1)	not complete
NO (-6)	channel not open

Description

This procedure will pan the whole of the current cursor line in the specified channel window. A pan to the left can be obtained by specifying a negative distance. The distance to pan is always specified in terms of pixels. Vacated pixel positions will be filled with the 'paper' colour.

The height of the cursor line will depend upon the character font size (i.e., either 10 or 20 pixel rows). The cursor position will not be altered.

SD.PANRT $1F (31)

Pan RHS of cursor line

Entry parameters:	D1.W	Distance to pan
	D3.W	Timeout
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1, A1	
Additional errors:	NC (-1) not complete	
	NO (-6) channel not open	

Description

This procedure will pan the whole of the right-hand side of the current cursor line in the specified channel window. A pan to the left can be obtained by specifying a negative distance. The distance to pan is always specified in terms of pixels. Vacated pixel positions will be filled with the 'paper' colour.

The height of the cursor line will depend upon the character font size (i.e., either 10 or 20 pixel rows). The right-hand end includes the character at the current cursor position. The cursor position will not be altered.