1 Introduction

The screen driver handles the display of any number of video "pages" in the various different display modes which the video chip provides although it does not support all the possible modes.

The display is managed in terms of video "pages", with one page corresponding to each channel. Before a channel is opened to the screen driver the user must specify various parameters, such as a video mode and page size, by setting EXOS variables. A channel can then be opened to device **"VIDEO:"**. If a filename or unit number is specified then it will be ignored. The video driver will work out how much screen RAM it needs for this video page and obtain that much RAM from EXOS, including enough for the various variables needed.

Once the channel has been set up in this way, the user can read and write characters or blocks of data. This data will be interpreted differently by pages of different modes, particularly control characters and escape sequences.

At this stage however, the video page will not be visible on the display. A special function call is required to cause a video page to be actually displayed on the screen. It is only at this time that the appropriate line-parameter blocks are set up and the text/graphics will appear. It is possible to display any vertical section of a video page at any vertical position on the screen, covering up anything which was displayed on those scan lines before. If the page width is less than the full screen width then the margins will be adjusted to display the page in the middle of the screen.

The screen driver has a 256 character font in video RAM which it uses for all character type displays. This is initialised to a standard ASCII character set repeated twice but any character may be re-defined by the user. Each character is 8 pixels wide and 9 lines deep. These values include the space between characters and between lines.

1.1 Co-ordinate Systems

The co-ordinate system used in specifying graphic positions etc is standardised so that giving the same commands to two pages of different resolutions or colour modes will produce a pattern of the same size on the screen. A graphics page of full screen size will be 972 logical pixels high and 1344 pixels wide. This corresponds to twice the maximum horrizontal and four times the vertical resolution available. All beam positions etc. are specified in these co-ordinates, and depending on the colour mode the actual position will have to be an approximation.

Text pages do not use this co-ordinate system, they use a system based on character positions so the top left corner is (1,1) and the top right corner (of a full screen size low resolution text page) is (1,42).

Attribute graphics pages actually keep a beam position in graphics co-ordinates and a seperate cursor position in text co-ordinates. The use of these is explained later.

2 Basic Control of Video Pages

As mentioned before, each video page is a seperate channel. When a channel is opened to the video driver this implies that another video page is to be created. The video driver looks at EXOS variables which specify the page size, page mode and colour mode. These variables must be set up by the user before opening a video channel. From these variables the video driver determines how much video RAM it needs and obtains that much with an EXOS function call ("Allocate channel buffer").

The video driver maintains the line parameter table in a fixed place in its absolute device RAM area. The line parameter table always consists of 28 line paraneter blocks of 9 scan lines each for the display area and various other ones to generate the frame sync and borders. The first line parameter block is reserved for the status line display which is a fixed area of RAM. The other 27 line parameter blocks can display any part of any page, so display is always in vertical units of 9 pixels. All 28 line parameter blocks are initially set up to be blank (ie all border colour). The variable LP_POINTER in the EXOS variable area points to the start of the line parameter table.

2.1 Display Modes

The display mode is specified by an EXOS variable MODE_VID the allowed values of which are:

```
 0  -  Hardware  text  mode (up to 42 chars/line).
 1  -  High resolution pixel graphics.
 2  -  Software text mode (up to 84 chars/line).
 5  -  Low resolution pixel graphics.
15  -  Attribute graphics.
```

The three graphics modes correspond to the PIXEL, LPIXEL and ATTRIBUTE modes of the Nick chip (see separate Nick chip specification).

2.2 Colour Modes

As well as the display mode, each video page is of a particular colour mode. The colour mode is specified by an EXOS variable called COLR_VID. The allowed values for this variable are:

0 - Two colour mode
1 - Four colour mode
2 - Sixteen colour mode
3 - 256 colour mode

For text modes it is only useful to use two colour mode, unless the characters in the font are re-defined for doing some sort of block graphics. Also attribute mode must always be in two colour mode, although sixteen colours will actually be available.

2.3 Page Size

Two EXOS variables, X_SIZ_VID any Y_SIZ_VID, define the size of the page to be created. The vertical size is specified in character rows. It can be any value from 1 to 255 although only 27 rows can be displayed on the screen at one time. The horizontal size is specified in low resolution character widths, and can be any number from 1 to 42.

A special function call is provided to return the size of a video page. It returns the number of lines and the number of characters per line. The characters per line value returned is the actual number of characters per line so in the case of a software-text mode it will be double the value in X_SIZ_VID when the channel was opened.

The parameters for this call are:

Parameters:	A = Channel number (1...255)
	B = 2 (Special function code)
Returns:	A = Status
	B = Number of characters per row
	C = Number of rows
	D = Mode of page (0, 1 or 2, 5 or 15)

2.4 Display Control

Video pages are not actually displayed on the screen until the user explicitly requests this. This request is done by a special function call. The parameters for this call are:

Parameters:
- A = Channel number (1...255)
- B = 1 (Special function code)
- C = First row in video page to display (1...size)
- D = Number of rows to display (1...27)
- E = row on screen where first row should display (1...27).

Returns:
- A - Status

The three row parameters are all given in character row units since the area of screen specified must be a whole number of line parameter blocks. The displayed page will replace anything which was displayed on that part of the screen before. If the channel is subsequently closed then any part of the screen which was displaying that channel will be made border colour (by bringing the margins in the relevent line parameter blocks right in).

A value of 1 for the position on screen parameter (given in register E) refers to the line on the screen directly below the status line. Thus it is not possible to overlay the status line since zero will not be accepted.

If a value of zero is given for the position in the page parameter (register C) then the portion of the screen defined by the other two parameters will be blanked (ie. made entirely border colour).

3. Character Output

The screen driver supports both the single character write and the block write EXOS function calls. A block write is exactly equivalent to writing all the characters individually, except that it is rather faster as it avoids the overhead of going through EXOS for every character. Block write is implemented using the general purpose WRBLOCK utility routin.

3.1 Printing Characters

All characters above 1Fh will be treated as printing characters and will be put at the appropriate place on the video page. All modes have some sort of "cursor" which moves when a charcater is printed but the details vary between different modes.

The bit maps for characters are stored in a fixed character font which is initialised to an ASCII character set. Each character is eight bits wide and nine bytes deep. The user can re-define any of these characters by an

escape sequence as specified below.

3.1.1 Text Mode Character Printing

Text pages (modes 0 and 2) maintain a single text cursor which in text co-ordinates. The printing character is displayed at this position and the cursor moved to the next character slot. At the end of a line the cursor automatically moves to the start of the next line, with automatic scrolling if it is at the bottom of the screen (this automatic scrolling can be disabled).

3.1.2 Pixel Graphics Mode Character Printing

Pixel graphics pages maintain a beam pointer in graphics co-ordinates. The printing character is put at this beam position and the beam moved to the next character position, moving to the start of the next line if at the end of a line. Characters can be put at any pixel position, not just on character boundaries. There is no scrolling. If the beam is too near the bottom of the page to fit the character on then it will not print anything.

Characters are printed in the current ink colour regardless of whether the beam is on or off and the paper colour of the charcater is unaffected. The characters are plotted by reading bits out of the font and plotting the pixel in the current ink colour if the bit is set. Thus the characters will be correct in any colour mode. To improve legibility the character height is doubled for sixteen and 256 colour mode.

3.1.3 Attribute Graphics Mode Character Printing

Attribute mode character printing is rather more complex. All attribute graphics pages maintain a seperate text cursor in text co-ordinates and a graphics beam position in graphics co-ordinates. Characters are printed at the text cursor position and so will always be in exact character positions. At the end of a line it will go on to the start of the next line and at the end of the page it will go back to the top left of the page - there is no scrolling.

The character is displayed by copying the currently selected ink and paper colours into all attribute bytes corresponding to this character (nine of them). The character data itself from the font is then put into the pixel data area but only if the beam is on. If the beam is off the pixel data is unaffected, only the attrbute data is affected. This allows the colours of part of an existing display to be changed.

3.2 Control Codes and Escape Sequences.

Character in the range 00h to 1Fh are control characters and are not printed. Some of these are interpreted by video pages, depending on the mode. Any which are not understood are simply ignored. A special control code is ESCAPE (ASCII 1Bh) which is used to start an escape sequence for carrying out various functions.

Here is a list of the control codes and escape sequences interpreted by the various modes. The more complex ones of these are explained further in the following sections.

3.2.1 Codes Interpreted by Any Video Page

^Z (1Ah) - Clear entire page and home cursor/beam.

^J (0Ah) - Line-feed. Move cursor down to next line (scrolls if at bottom of screen in text mode and scroll is enabled.)

^M (0Dh) - Carriage return. Returns cursor to start of current line

^^ (1Eh) - Cursor/beam home. (ASCII RS)

escK - Define character (see below)

escC - Set all palette colours \ see below for parameters.
escc - Set one palette colour /

escI<n> - Set ink colour to <n> \ See below for details of
escP<n> - Set paper colour to <n> / these in different modes

esc=<y><x> - Set cursor position (see below)

3.2.2 Codes Interpreted by Text Pages Only

^Y (19h) - Clear to end of line. Does not move cursor.

^H (08h) - Cursor left. (ASCII BS)
^I (09h) - Cursor right. (ASCII TAB)
^K (0Bh) - Cursor up. (ASCII VT)
^V (16h) - Cursor down. (ASCII SYN)

esc? - Read cursor position. Also supported in attribute mode. (see below)

esc.<n> - Set cursor character to character code <n>.

escM<n> - Set cursor to palette colour <n>

escO - Set cursor on.
esco - Set cursor off.

escS - Set automatic scroll on
escs - Set automatic scroll off

escU<m><n> - Scroll up lines (m-20h) to (n-20h). m <= n
escD<m><n> - Scroll down lines (m-20h) to (n-20h). m <= n

3.2.3 Codes Interpreted by Graphics Pages Only

escA<xx><yy> - Position beam at co-ordinates (xx,yy) where xx & yy are each 16-bit hex numbers specified low byte first.

escR<xx><yy> - Relative beam movement by amount (xx,yy).

esc@ - Read beam position. (see below)

escS - Set beam on.
escs - Set beam off.

esc.<n> - Set beam to line style <n> - see below.
escM<n> - Set beam to line mode <n> - see below.

escF - Graphics fill - see below.
escE - Plot ellipse - see below.

3.3 Position Cursor

The escape sequence to position cursor works in all modes. In text mode it simply moves the cursor. In attribute graphics mode it moves the text cursor but leaves the graphics beam pointer alone. In pixel graphics modes it moves the graphics beam pointer to the appropriate text co-ordinates, so it will be on a character boundary.

The format of the escape sequence is:

esc=<y><x>

This sets the cursor to row (y-20h) and column (x-20h). If either <x> or <y> is 20h (thus setting row or column zero) then that co-ordinate will remain un-changed. This allows just the row or column to be set.

3.4 Define Character

This escape sequence allows the user to re-define one of the 256 characters. Although it is sent to a specific channel, it actually affects a global character font and will thus affect other channels. The syntax of the escape sequence is:

escK<n><r1><r2><r3><r4><r5><r6><r7><r8><r9>

where: <n> is the character number (0...255)
<r1>...<r9> are the bytes for the nine rows of the character. <r1> is the top row.

Note: In high resolution text mode, only the middle six bits of the character bytes will actually be displayed as the other two are masked out and used to control the colour selection.

3.5 Palette Colours

Each video page has a palette of eight colours associated with it which will be initialised to some useful set of colours (such as black, red, green, yellow, blue, magenta, cyan, white). There is an escape sequence with which the user can change all these colours. the format of this is:

escC<c><c><c><c><c><c><c><c>

Each <c> is a byte specifying one of the palette colours and there must always be eight of them.

There is another escape sequence which allows just one palette colour to be changed. The format of this is:

escc<n><c>

Where <n> is the palette colour number 0...7 and <c> is the new value for this palette colour.

When new palette colours are selected any line parameter blocks which correspond to this video page will be updated.

3.6 Ink and Paper Colours

The user may specify a palette colour for both the ink and paper colour with seperate escape sequences. For all video modes the ink colour defaults to one and the paper colour to zero.

For pixel graphics pages the allowed ink and paper colours depend on the colour mode so it is 0 or 1 in two colour mode, 0...3 in four colour mode, 0...15 in sixteen colour mode and 0...255 in 256 colour mode. Pixels are always plotted in the current ink colour. The paper colour of the display is only changed when the page is cleared.

For attribute graphics mode the ink and paper colours can be in the range 0...15 and they control what colours are put into the attribute bytes. When a character is plotted both the ink and paper colours of the relavent attribute bytes will be set up. When plotting lines, ellipses or filling however, only one of them is set up depending on the line mode (see later for details). The line mode also controls whether ones or zeroes are put into the pixel data area.

In four, sixteen or 256 colour text modes the ink and paper colours have no effect, the palette colours for each pixel are determined directly from bits in the character font.

In two colour hardware text mode the ink and paper colours are always (0,1) or (2,3). These interact with the top bit of the character number being printed. Basically the top bit is complemented if colour pair (2,3) is selected.

In two colour software text mode four colour pairs are available, (1,0), (2,3), (4,5) and (6,7). Characters are always printed in the current colour pair. There is no interaction with the character codes.

3.7 Graphics Line Style

For a graphics page the line-style may be set with an appropriate escape sequence. This specifies a single byte has the following meanings:

1	-	Solid line (default)
2..14	-	Various types of broken and dotted lines.

3.8 Graphics Line Mode

An escape sequence specifies the line mode byte which has the following meanings:

0	-	PUT plotting (default)
1	-	OR plotting
2	-	AND plotting
3	-	XOR plotting

For pixel graphics pages when plotting a pixel the current ink colour is combined with the old colour of the pixel according to the operation selected by the line mode and then stored.

For attribute graphics pages the line mode is more complex. Line modes 4 to 7 are also defined which are equivalent to 0 to 3 but do "paper" plotting rather than "ink" plotting. When paper plotting is selected the current paper colour is put into the attribute byte, leaving the ink colour in this byte unaffected. When ink plotting is seected just the ink colour is put into the attribute byte, leaving the paper colour unaffected. In adition the plotting on the pixel map will use a one bit for ink plotting and a zero bit for paper plotting. This bit will be combined with the old bit for this pixel according to the PUT/OR/AND/XOR mode and the result stored.

3.9 Graphics Fill - Paint

The graphics fill command is a simple escape sequence which does a fill from the current beam position. It fills in the current ink colour up to any boundry which is not the same colour as the current beam position. It handles concave shapes and tests for reaching the edge of the video page. It may fail to fill the entire shape if it runs out of stack but this should only happen with extremely complex shapes since it does garbage collection on the stack when it gets full.

3.10 Graphics Ellipse Drawing

The ellipse drawing routine takes two 16-bit parameters (low byte first) specifying the x and y radii. To draw a circle these should be the same value. The centre of the ellipse will be at the current beam position. The format of the escape sequence is:

escE<xx><yy>

3.11 Border and Fixed Bias Colours

Two EXOS variables BORD_VID and BIAS_VID are provided to control the hardware border and fixed colour bias registers. The values in these variables are written out to the NICK chip on every interrupt. The border colour is written directly to the border register. The top 5 bits of the fixed bias variable are written to the bottom 5 bits of the fixed bias register, and the top bit of the register is set according to the EROS variable MUTE_SND since it is used to silence the internal speaker.

4. Character Input

4.1 Simple charcater input

When a read character (or one character of a read block) is done from the video driver the result depends on the mode. In text modes the ASCII code of the character at the cursor position will be returned, without moving the cursor. With graphics mode the palette colour of the pixel at the current beam position will be returned. This will be 0 or 1 in two colour mode, 0...3 in four colour mode, 1...15 if sixteen colour or attribute modes and 0...255 in 256 colour mode.

4.1 Reading Cursor Position

The escape sequence: esc? is supported in text and attribute modes. This triggers the video channel to return the current cursor co-ordinates as the next two characters read from this channel. The co-ordinates will be returned in the same way as they are specified for cursor positioning, ie. with a 20h offset added on.

4.2 Reading beam position

This is supported by graphics pages only (including attribute mode). The escape sequence: esc@ will trigger the channel to return the current graphics beam position as the next four bytes read from the video page. The co-ordinates are returned in the same format as they would be specified for an absolute beam position.

5. Status Line

As mentioned earlier there is a special status line display which is outide the normal page/channel structure. The first line parameter block is reserved for this status line and will never be used to display any other page. The two byte variable LP_POINTER which is at a fixed address in the EXOS variable area contains the address of the start of this line parameter block. This is used by the cassette driver to modify the palette colours of the status line to provide the cassette level meter display.

An EXOS variable (called ST_FLAG) is provided which if zero will cause the status line to be displayed and if non-zero will cause it to be un-displayed (that region of the screen will be border colour). This is implemented by the video driver examining ST_FLAG every interrupt and setting the margins in the reserved line parameter block appropriately.

There is a two byte pointer (ST_POINTER) defined at a fixed address in EXOS variable area which contains the address of the 40 byte area of RAM which is the status line. This pointer will point to Z-80 page-2 and the RAM will be at this address in segment 0FFh. The user or external devices can access the status line by finding its address from ST_POINTER. Built in devices can access it directly by using the public symbol ST_LINE which is the start of the status line RAM (in Z-80 page-2 segment 0FFh).

