4 The Text VDU.

The Text VDU is a character based screen driver. It controls 8 different streams each of which can have an area of screen allocated to it (a window). The Text VDU allows characters to be written to the screen and read from the screen. It also treats certain 'characters' as 'control codes' which can have various effects, from moving the cursor to setting the colour of an ink.

4.1 Text VDU Coordinate Systems.

The Text VDU uses two coordinate systems - logical and physical. Generally the user specifies positions to the Text VDU in logical coordinates. Physical coordinates are used internally and occasionally by the user to specify positions to the Text VDU. Both systems use signed 8 bit numbers and work in character positions. Each character position is 8 pixels (dots) wide and 8 pixels high. This means that the position of a coordinate on the screen depends upon the screen mode.

Physical coordinates have columns running left to right and rows running top to bottom. The character position at the top left corner of the screen is row 0, column 0.

Logical coordinates are similar to physical coordinates except that the character position at the top left corner of the text window is row 1, column 1.

4.2 Streams.

The Text VDU has facilities for handling up to 8 streams at once. Each stream has an independent state (although some facilities are shared and thus affect all streams when altered). The features that are stream dependent are:

VDU enable.
Cursor enable (enable or disable, on or off).
Cursor position.
Window size.
Pen and paper inks.
Character write mode (opaque or transparent).
Graphics character write mode.

The features that affect all streams include:

Character matrices.
Control code buffer.
Text VDU indirections.
Screen mode.

All these features are explained in detail in the sections below.

At any time, the stream which is currently selected may be changed without adverse effects provided that the control code buffer is not in use (see section 4.7 for further explanation). A stream will remain selected until another stream is selected. This means that a program need not know which stream it is using.

The default stream, selected at EMS, is stream 0.

BASIC extends the stream concept to include the printer and cassette/disc files. This extension is not part of the firmware.

4.3 Text Pen and Paper Inks.

Each stream has a pen and a paper ink associated with it. The text pen ink is used to set the foreground pixels in characters (see section 4.6). The text paper is used to set the background pixels in characters and to clear the text window.

The pens and papers can be set to any ink that is valid in the current screen mode (see section 6. 1). The default setting for a stream has the paper set to ink 0 and the pen set to ink 1. Changing a pen or paper ink does not change the screen; it merely alters how characters will be written in the future.

4.4 Text Windows.

Each stream has a text window associated with it. This window specifies the area of the screen where the stream is permitted to write characters. This allows different streams to use different portions of the screen without interfering with each other.

Windows are trimmed so that they fit within the current screen (whose size varies with the screen mode, see section 6.1). The smallest size window allowed is 1 character wide and 1 character high.

Before writing to the screen the position to write at is forced to lie inside the window (see 4.5 below). This may cause the window to roll. Other operations, such as obeying certain control codes also cause the write position to be forced inside the window.

A text window which does not cover the whole screen is rolled by the firmware copying areas of screen memory around. There is no alternate method available. This makes rolling large windows a fairly time consuming process.

A text window which covers the whole screen is rolled by using the hardware rather than by copying areas of memory. The offset of the start of the screen in the screen memory can be set (see section 6.4). By changing this offset by + 80 or - 80 the whole screen can be rolled up or down by a line of characters.

It is obviously a good idea to prevent windows that are being used from overlapping. If they are allowed to overlap then the portion in multiple use will merely contain whatever was written to it last. There is no precedence of windows one over another. A window occupying the whole screen will overlap the other windows and so if this window is rolled it will move the contents of the other windows.

The default windows, set up at EMS and after changing screen mode, cover the whole of the screen. All eight windows overlap.

4.5 The Current Position and the Cursor.

Each stream has a current position associated with it. This is where the next character to be printed on the screen is expected to be placed. However, if, when a character is to be printed, the current position is found to lie outside the text window then it is forced inside. The following steps are applied in turn to force the current position inside the window:

1/ If the current position is left of the left edge of the window then it is moved to the right edge of the window and up one line.

2/ If the current position is right of the right edge of the window then it is moved to the left edge of the window and down one line.

3/ If the current position is now above the top line of the window then it is moved to the top line of the window and the contents of the window are rolled down one line.

4/ If the current position is now below the bottom line of the window then it is moved to the bottom line of the window and the contents of the window are rolled up one line.

When the cursor is enabled, the current position is marked by the cursor blob. However, before placing the cursor blob on the screen, the current position is forced to lie inside the current window just as it is before a character is placed on the screen. This may cause the current position to move.

If the cursor is disabled then the current position may lie outside the window and it will not be forced inside the window until, for example, a character is printed.

The current position can be changed directly (by calling TXT SET CURSOR, TXT SET ROW or TXT SET COLUMN) or by sending control codes to the Text VDU. The location the current position is moved to is not forced inside the window immediately, but only when the window is to be written to, as described above. This allows the current position to be changed by moving via a position outside the window, if required.

There are two ways to disable the cursor and prevent the cursor blob from appearing on the screen. The first, cursor on off, is intended for use by system programs. This is used by BASIC, for example, to hide the cursor unless input is expected. The second, cursor enable disable, is intended for use by the user. The cursor blob will only be placed on the screen if it is both on and enabled.

In V1.1 firmware it is possible to interrogate the current enable disable states of the VDU and cursor for the current stream using TXT ASK STATE.

The cursor blob is normally an inverse patch. The character at the cursor position is displayed with the text pen and paper inks reversed. This makes it easy to restore the original form of the character position if the cursor is moved. It is possible for the user to alter the form of the cursor blob, if required, by changing the indirections TXT DRAW CURSOR and TXT UNDRAW CURSOR.

4.6 Characters and Matrices.

A character is displayed on the screen in an area 8 pixels (dots on the monitor) wide and 8 pixels high. Thus the maximum number of characters on the screen depends upon the screen mode, (see section 6.1). Each character has a matrix which is an 8 byte vector that specifies the shape of the character. The first byte of the vector refers to the top line of the character and the last byte to the bottom line of the character. The most significant bit of a byte in the vector refers to the leftmost pixel on a line of the character and the least significant bit refers to the rightmost pixel on a line of the character. If a bit in the matrix is set then the pixel is in the foreground. If a bit is clear then the pixel is in the background.

A foreground pixel in the character is always set to the pen ink. The treatment of a background pixel depends on the character write mode of the VDU. In the default mode, opaque mode, background pixels are set to the paper ink. There is another mode, transparent mode, in which the background pixels are not altered. Thus, in transparent mode, the character is written over the top of the current contents of the screen. This is useful for annotating pictures or generating composite characters.

The Text VDU is capable of printing 256 different characters, although special effort is required to print the first 32 characters which are usually interpreted as control codes. The matrices for the characters are normally stored in the ROM but the user may arrange for any number of the characters to have matrices stored in RAM where they may then be altered. The default setting, at EMS, is to have all the matrices in ROM. (BASIC takes special action during its own initialization to create 16 'user defined' matrices.) The default character set is described in Appendix VI.

When the user sets up a table of user defined matrices, by calling TXT SET M TABLE, it is initialized with the current settings of the matrices from ROM or RAM. This means that extending the table does not alter the current matrices. Contracting the table will make the characters lost revert to their default matrices in ROM.

When characters are read from the screen (by calling TXT RD CHAR) the pixels on the screen are converted to the form of a matrix. This is compared with the current character matrices to find which character it is. This means that changing the character matrices or altering the screen may make a character unrecognisable, in particular, changing the pen or paper ink can cause confusion. Usually these problems. result in the character appearing to be a space (character #20) and so special precautions are taken to avoid generating spaces - after some ink changes real spaces may be read as block graphic characters # 80 or #SF.

To allow the user to change how characters are written to and read from the screen, the indirections TXT WRITE CHAR and TXT UNWRITE are provided.

4.7 Character Output and Control Codes.

The main character output routine for the Text VDU is TXT OUTPUT. This obeys controls codes (characters 0-31) and prints all other characters. Characters sent to TXT OUTPUT pass through various levels of indirection and can be dealt with by various output routines.

TXT OUTPUT uses the TXT OUT ACTION indirection to sort out whether the character is a printing character, is a control code to be obeyed or is the parameter of a control code.

TXT OUT ACTION normally calls TXT WRITE CHAR to print characters on the screen. However, if the graphic character write mode is selected then characters are printed using the Graphics VDU character write routine (see 5.6 below). This mode can be selected on a character by character basis using a control code or on all characters sent (see TXT SET GRAPHIC). When graphic character write mode is selected control codes are not obeyed but are printed by the graphics routine instead.

TXT OUT ACTION deals with a control code in the following manner:

1/ The code is stored at the start of the control code buffer.

2/ The code is looked up in the control code table to find out how many parameters it requires.

3/ If no parameters are required go directly to step 5.

4/ If one or more parameters are required then TXT OUT ACTION returns but the next characters sent to it are added to the control code buffer rather than being printed or obeyed. This continues until sufficient parameter characters have been received.

5/ The code is looked up in the control code table to get the address of the routine to call to perform the control code and this routine is then executed.

6/ The control code buffer is discarded and the next character sent may be printed or may be the start of a new control code sequence.

The user can change the operation of a control code by changing the entry for it in the control code table (see TXT GET CONTROLS). This contains a 3 byte entry for each code and entries are stored in ascending order (i.e. the entry for #00 first, #01 next and so on).

Bits 0-3 of the first byte of each entry specifies the number of parameters required. This must lie in the range 0-9 as the control code buffer is only capable of storing up to 9 parameters.

In V1.1 firmware bit 7 specifies whether the code is affected when the VDU is disabled. If bit 7 is set then the code is to be ignored when the VDU is disabled, otherwise it is to be obeyed.

The second and third bytes are the address of the routine to call to obey the code. This routine should lie in the central 32K of RAM or in the lower ROM (which will be enabled). It should conform to the following entry/exit conditions:

Entry:

A contains the last character added to the buffer.

B contains the number of characters in the buffer (including the control code).

C contains the same as A.

HL contains the address of the control code buffer (points at the control code).

Exit:

AF, BC, DE and HL corrupt.
All other registers preserved.

The control code buffer is shared between all the streams. A control code sequence should be completed before the stream is changed otherwise unexpected effects may occur.

The default control code actions, set at EMS and when TXT RESET is called, are described in Appendix VII.

It is possible to disable a text stream by calling TXT VDU DISABLE. When disabled the stream will not write any characters to the screen and in V1. l firmware control codes may not be obeyed (as described above). Normal operation can be restored by calling TXT VDU ENABLE. Note, however, that calling these routines will empty the control code buffer. This effect may be used to avoid problems when the state of the control buffer is unknown (when printing an error message perhaps).