1. General Device Interface

The printer driver is a very simple device which just sends characters to a printer (or other device) using the built in centronics type parallel interface.

Only one channel at a time may be open to the printer driver, if an attempt is made to open a second channel then an error (.2NDCH) will be returned. A channel can be opened by giving the device name "PRINTER:", any filename or unit number is ignored.

Having opened a channel characters can be written using either the single characer write or the block write function call. The characters will be sent without any interpretation at all, and all 8 bits are sent.

2. Hardware Details

The hardware consists of one eight bit output port for the parallel data (port 0B6h), one other output bit for a data strobe (bit 4 of port 0B5h) and one input bit as a ready signal (bit 3 of port 0B6h).

To send a byte the printer driver outputs the character to the data port and then waits until the ready signal goes low. When the ready signal is low it strobes the data by setting the data strobe low for a few microseconds and then setting it high again (it is normally high when not in use). This completes the sending of a character.

The other bits of output port 0B5h are used for various control operations such as scanning the keyboard, and controlling remote control relays. A variable (PORTB5) which is at a fixed address defines the current state of this port and the printer driver ensures that all other bits of the port are maintained in their correct state.

3. Quick Reference Summary - EXOS calls

OPEN/CREATE CHANNEL	-	Treated identically. Only one channel. Device name "PRINTER:". Filename and unit number ignored. No EXOS variables to be set before open.
CLOSE/DESTROY CHANNEL	-	Treated identically.
READ CHARACTER/BLOCK	-	Not supported.
WRITE CHARACTER/BLOCK	-	Writes bytes without interpretation.
READ STATUS	-	Not supported.
SET STATUS	-	Not supported.
SPECIAL FUNCTION	-	No special functions.

++++++++++ END OF DOCUMENT ++++++++++

1. Introduction

All the other built in device drivers provide an interface to some aspect of the hardware such as the cassette I/O circuitry or the DAVE chip. The editor however does not interface directly to any hardware, instead it provides a higher level user interface to two of the other built in drivers - the video driver and the keyboard driver.

An editor channel can be thought of as an intelligent, full screen editing terminal handler. It can be used by an applications program to provide all of its general purpose communication with a user. For example the IS-BASIC cartridge does all of its screen and keyboard I/O through an editor channel. BASIC will be used frequently in this document as an example of how to use the editor.

The editor can support any number of channels open to it at a time, each channel corresponds to a separate "document" which is being edited. The word document here is used loosely since for example the editor channel used by BASIC is referred to as a document although it is actually a collection of BASIC commands, program listings, error messages, program output, etc.

Each editor channel has video channel and a keyboard channel associated with it. Different editor channels can share the same keyboard channel (which is essential since the keyboard driver only allows one channel to be open to it), but must have separate video channels.

Each editor channel also has an area of channel RAM which it uses for a text buffer. This buffer can be any size from a few hundred bytes to just under 16k and will typically be a few kilobytes. Text can be entered into the editor's buffer either from the applications program or from the keyboard. The editor writes characters to the video page in such a way that it is kept updated to form a "window" onto the text buffer. This is not a true window since the video page has its own copy of the text it is displaying.

2. Opening Channels

An editor channel can be opened by giving the device name "EDITOR:", any filename or unit number is ignored. Before opening an editor channel, three EXOS variables must be set up. These are:

VID_EDIT	-	Channel number of video page.
KEY_EDIT	-	Channel number of keyboard channel.
BUF_EDIT	-	Size of editor buffer in units of 256 bytes.

The video and keyboard channels specified in VID_EDIT and KEY_EDIT must be opened before opening the editor channel. The video page must be a text mode and must be at least 3 rows by 4 characters. The editor determines the size and mode of the video page when the channel is opened and returns an error (.EVID) if it is unsuitable. Note that the editor does not display the video page on the screen, it is up to the applications program to take care of this.

The actual size of the editor's buffer which is available for storing text is:

256*BUF_EDIT + n

where 'n' is between zero and 255 and depends on the width of the video page (space reserved for the ruler line) and the exact size of the editor's variable area. The valid range for BUF_EDIT is thus zero to 254.

The editor will work with any size buffer but it is sensible to ensure that it is at least as big as the video page so that the editor is always capable of displaying a full page. The editor stores lines as variable length in its buffer so, since short lines are common, it generally manages to store more than the calculated minimum number of lines.

3. General Editor Features

3.1 The Editor's Text Buffer

As mentioned before the editor has a text buffer in which it stores its text, and the video page just provides a window onto part of this buffer. Text is stored in the buffer on a line orientated basis. Each line has a three byte line header containing certain flags and margin information. This is followed by the text of the line itself stored in ASCII. The line is terminated by a special character which indicates whether it is the last line in a paragraph and whether it is the last line in the buffer.

The lines are stored with variable length so if a line only has four characters on it, followed by 36 spaces then the 36 spaces are not stored. This improves buffer usage very significantly since in general short lines are quite common. There is no limit to the number of lines in the buffer other than the total size of the buffer.

The buffer is arranged as a circular buffer so the start of the text may be anywhere in the buffer, and the end of the buffer may occur at any point in the text with the text continuing again at the start of the buffer. This avoids ever having to move the whole text up or down in the buffer.

3.2 When the Buffer Becomes Full

When new text is entered into the buffer, either at the end or into the middle of the buffer this clearly uses up buffer space. Eventually the buffer may become full. In fact it is generally the case when using BASIC that the buffer is nearly full most of the time, as it contains previous commands and so on which have scrolled off the screen.

Whenever there are less than 100 bytes spare in the buffer the editor displays a number on the right hand side of the status line indicating the number of free bytes. As characters are typed in, this number will get smaller until it eventually reaches zero. This number is only displayed when waiting for keyboard input from the user, so for example the number cannot be seen when BASIC is listing a program even though the buffer may be full.

When the buffer is full and another character is typed in (or written by the applications program), the editor has to delete some of the existing text to make room. It always deletes a whole line of text and it generally deletes the first line since this will normally be the oldest and least useful one. If the first line of the text is displayed on the video page then it deletes the last line instead, to avoid deleting text which is displayed.

If the editor buffer is very small or there are some very long lines, then every line may be displayed at once, so the editor has no choice but to delete a line which is displayed. In this case the editor deletes the last line unless the cursor is on the last line, in which case it deletes the first line and scrolls the page up.

Note that because the editor buffer is circular, deleting this line of text does not involve moving the whole text up to fill the gap (at least not usually).

3.3 Margins and the Ruler Line

Each line in the buffer has its own individual left margin position. When a new line is created it will be given a left margin equal to the current left margin setting which can be displayed on a ruler line at the top of the video page. There is also a right margin which is displayed on the ruler line. In general text can only be entered in between these margin settings although there is the facility of temporarily releasing the margins.

3.4 Paragraphs

Lines in the editor's buffer are grouped together in paragraphs. When the user presses ENTER (or a CR is received from the applications program) this marks the current line as the end of a paragraph. It also moves the cursor to the start of the next line which will be the start of a new paragraph (and may have the side effect of sending text to the applications program - see later).

If the user types a very long line then the editor will split the line at a sensible point (using a process called word wrap described in the next section) to give two lines. The first line will be terminated by a soft carriage return marker to indicate that it is not the end of a paragraph. In this way long paragraphs can be built up.

There is no indication on the screen of where paragraphs start and end but some of the editing functions operate on paragraphs, and the paragraph is the basic unit for sending text back to the applications program.

3.4 Word Wrap

Word wrap is the process which decides where to split a line which is too long. When a character is typed outside the margins (assuming that margins are not released) then the editor searches back to find the start of the word which contained that character and moves the whole of that word onto the start of a new line.

This process is done with all text received from the applications program as well as that typed at the keyboard from the user. Thus BASIC listings are subject to word wrap so keywords and variable names etc. will not be split in half.

3.5 Long Lines

Although a very long line which is typed in will be split by word wrap, it is possible to create a long line by inserting characters into the middle of a line, which will push the rest of the line to the right. In this way it is possible to create a line which is too long to be all displayed on the video page. This fact is marked by a red angle bracket (">") on the extreme right hand end of the line on the video page. This is an overflow marker.

The part of the line which has gone off the page cannot be accessed although it is remembered by the editor. The line can only be accessed by reformatting it to bring it back onto the page. There are various editing commands which can do this described later on.

3.6 Flashing Cursor

The EXOS video driver which the editor uses only provides a static non-flashing cursor display, although this can be turned on and off. The editor implements a flashing cursor by simply turning the video page's cursor on and off regularly while it is waiting for input from the user. It always ensures that the cursor is switched off when it is doing any editing functions, since these can result in the cursor having to move all over the screen and it is rather messy if the cursor can be seen doing this.

The editor also switches the cursor off whenever it returns to the applications program and it remains off when the applications program is writing characters to the editor. This results in a nice clean cursor display, where the flashing cursor always means that the editor is waiting for the user to type some input.

4. Writing to the Editor

Any characters in the range 20h to 9Fh, written to the editor are regarded as printing characters and are put into the text buffer at the current cursor position and displayed on the video page. They are subject to word wrap as described above.

The editor will also interpret the following control codes. All codes in the range 00h to 1Fh not mentioned here are ignored.

00h (NUL) - Writes a null to video to check it is still OK.
09h (TAB) - Move to, or insert spaces to, next tab stop.
0Ah (LF) - Ignored.
0Dh (CR) - Goes to start of new line (equivalent to CR-LF).
18h (^X) - Set left margin at cursor column.
19h (^Y) - Clear to end of line.
1Ah (^Z) - Clear whole buffer and screen and home cursor.
1Bh (ESC) - Starts escape sequence (see below)

The only escape sequence interpreted by the editor is to position the cursor at arbitrary co-ordinates. This is identical to the video driver escape sequence for this function and details can be found in the video driver specification. It positions the cursor at the specified co-ordinates of the video page, regardless of which portion of the editor's buffer is currently being displayed.

Codes in the range 0A0h to 0FFh are interpreted exactly as if they had been received from the keyboard. These provide various editing functions and cursor movement. They are described in detail in the section on editing functions.

5. Reading From the Editor

When the editor receives a read character function call, it examines the EXOS variable FLG_EDIT. This byte contains a series of flags which control the response of the editor to this read character call. The editor's action will first be described in general terms without reference to the individual flags. The effect of each flag will then be described in detail.

5.1 Basic Editor Read Action

Assuming for now a typical setting of the flags in FLG_EDIT, when the editor receives a read character call it interprets this as a request to send a line of text back to the applications program. It does not return a character immediately but records the state of the flags and enters its main editing loop which provides the actual editing facility to the user. It is only while in this loop that the cursor on the video page will flash. A flashing cursor thus indicates that the editor is waiting for a key to be pressed.

This loop reads a character from the keyboard, responds to it and then loops back for another one. This allows the user to type in text which will be inserted into the editor's text buffer and displayed on the video page, and to carry out various editing functions. The details of the editing functions are given later. There are two keys which are of importance here - ESCAPE and ENTER.

If ESCAPE (ASCII code 01Bh) is received from the keyboard then this character will immediately be returned to the applications program as the response to the read character call, regardless of the state of any of the FLG_EDIT flags. This provides a way of interrupting a line input operation.

If ENTER (ASCII code 0Dh) is pressed then this is a command to the editor to begin sending text back to the applications program. The details of how much text is sent are determined by the flags and will be described below. An internal editor flag is set to indicate that it is in the process of sending text, and the first character is returned to the applications program. When the applications program makes another read character call the internal flag is still set, so instead of entering the editing loop, it simply returns the next character of the requested text immediately. This continues until all the required text has been sent at which time the internal flag is cleared so the next read character call will again enter the editing loop.

It is up to the applications program to recognize when it has read all the characters to avoid re-starting a new read operation. How to recognize this depends on the setting of the editor flags and is described later.

Note that the editor flags are sampled once when the first read character call is made and then not again until all the text has been sent. Changing them in the meantime will therefore have no effect on the read which is in progress. If a write character call is mode while a read is in progress then the internal flag is cleared so that read will be aborted. This also applies if an special function calls are made.

5.2 The Editor Read Flags in Detail

The assignment of bits in the FLG_EDIT EXOS variable is:

```
        MSB  bit-7  -  SEND NOW
             bit-6  -  SEND ALL
             bit-5  -  NO READ
             bit-4  -  NO SOFT
             bit-3  -  NO PROMPT
             bit-2  -  AUTO ERA
             bit-1  -  not used
        LSB  bit-0  -  not used
```

5.2.1 The SEND NOW Flag (bit-7)

If this flag is set then the editor will start returning text immediately, without reading from the keyboard at all. If it is clear then the main editing loop will be entered and no text will be returned until the user types ENTER. In either case the amount of text returned is the same and depends on the setting of the other flags.

5.2.2 The SEND ALL Flag (bit-6)

This is the main flag which determines how much text is sent. If it is clear then the paragraph containing the cursor will be sent. If it is set then the whole editor buffer will be sent.

In the first case the applications program will be sent the characters of the paragraph one by one terminated with a CR and then an LF. This CR-LF can be used by the applications program to determine when to stop reading (beware if NO SOFT is clear! - see below). The cursor will be left on the first character of the next paragraph with a new (empty) paragraph being created if that was the last one.

If SEND ALL is set then the entire buffer will be sent. This will include CR-LF sequences at least between each paragraph (again see NO SOFT flag below) so this cannot be used to indicate the end of the text. Instead, after the last CR-LF has been sent (the last characters sent will always be CR-LF), the next character read will produce a .EOF (end of file) error. This error will only be received for one character so the applications program must notice it and stop reading. If another character was to be read then this would start the whole reading process again from the start of the buffer.

5.2.3 The NO READ Flag (bit-5)

If this flag is set then ENTER will not in fact return any characters at all to the applications program. The only way to get back to the applications program is thus to press ESCAPE. Although no characters are returned, the routine which selects which characters to send depending on the flags, is still executed. Thus the cursor is moved in the same way as if the text was returned. If SEND ALL is clear then pressing ENTER will just move to the start of the next paragraph, putting in a new line if it was at the end of the buffer. This will make the editor behave rather like a typewriter.

5.2.4 The NO SOFT Flag (bit-4)

This flag controls the sending of soft carriage returns and soft spaces. Soft carriage returns are those which separate successive lines of a paragraph. Soft spaces are those spaces which are inserted for justification, and also the spaces before the left margin of a line.

If this flag is clear then soft spaces are returned as normal ASCII spaces (20h) and soft carriage returns are returned as normal CR-LF sequences. If the flag is set then both of these are suppressed and no characters are returned for them.

Beware that if NO SOFT and SEND ALL are both clear then there is no way for the applications program to determine whether a CR-LF which it receives is the end of the paragraph, in which case it should stop reading, or simply the seperator between two lines of the paragraph, in which case it should continue. Therefore this combination of flags should be avoided - at least one of them should always be set.

5.2.5 The NO PROMPT Flag (bit-3)

This flag is normally clear. If it is set then the cursor position when the read operation was started is remembered. When it comes to returning text, if the cursor has not been moved out of the original paragraph, or to before the remembered position in the current paragraph, and if SEND ALL is clear, the the current paragraph will be sent back but starting from the character at the remembered cursor position rather than the start.

If the cursor has been moved out of these bounds then the whole paragraph will be returned as usual, or the whole editor buffer if SEND ALL is set.

This feature is used by BASIC when doing an INPUT command. It prints the prompt and the does an editor read with this flag set. When the paragraph is sent to BASIC the prompt will not be returned, just the response to it.

5.2.6 The AUTO ERA Flag (bit-2)

This flag is also normally clear. If it is set then, if the very first character typed at the keyboard is a printing character (as opposed to an editing function or cursor movement) then the current line will be cleared before responding to the key. This is provided mainly for BASIC to allow commands such as RUN to be typed on top of an existing line after editing a program. It may just conceivably be of some use to other applications programs.

5.3 Typical Flag Combinations.

The use of these flags can be rather confusing so this section discuses some examples of their use from IS-BASIC and the built in word processor (WP). This covers most useful combinations and certainly shows the use of each of the flags.

5.3.1 BASIC reading a command line. Flags = 00010lxx

When BASIC is reading a command line from the editor it is expecting either an immediate mode command (such as RUN) or a new line starting with a line number to by typed. In either case it wants to read a single paragraph entered by the user. This might be newly typed by him or it might be a line which already exists in the editor buffer which he simply moves the cursor to and re-enters.

Clearly BASIC wants to wait for the user to type ENTER so the SEND NOW flag is clear. Only one paragraph, rather than the whole buffer is wanted so SEND ALL is clear and BASIC wants to actually be sent the text so NO READ is clear. BASIC is not interested in the breaks between lines in the paragraph (since one command line can over flow onto several screen lines)_so NO SOFT is set. NO PROMPT is clear and AUTO ERA is set (as explained above).

5.3.2 BASIC doing an input command. Flags = 000110xx

When BASIC is doing an input command it also wants to read in a single paragraph so the SEND NOW, SEND ALL, NO READ and NO SOFT flags are all set the same as for reading a command. In this case however BASIC will write out a prompt and wants to read in just the response to that prompt, rather than having the prompt at the start of the paragraph which it receives. To do this it sets the NO PROMPT flag. The AUTO ERA flag is clear.

5.3.3 WP normal editing mode. Flags = 001xx0xx

In normal editing mode the word processor wants to let the user get on with his editing without data being sent to the word processor. The SEND NOW flag is clear to allow the user to do editing. The SEND ALL flag is clear and the NO READ flag is set to ensure that no characters are returned to the word processor when ENTER is pressed but the cursor will be moved to the start of the next paragraph. The NO SOFT and NO PROMPT flags are irrelevent when NO READ is set, and the AUTO ERA flag is clear.

This will have the effect of allowing the user to move all over his document, pressing ENTER and using any of the editing features. Only when ESCAPE is pressed will the word processor applications program be alerted. In fact the eight function keys have the ESC code as the first code in their programmed string so the word processor gets alerted when they are pressed as well.

5.3.4 WP Printing a Document. Flags = 11000xxx

When the word processor is asked to print a document it must read the whole of the editor's text buffer in order to print it. Also it wants to get the data immediately rather than waiting for the user to press ENTER. To achieve this SEND NOW and SEND ALL are both set. NO READ is of course clear or no text would be sent and NO SOFT is clear so that all spaces and soft carriage returns will be sent to ensure that the formatting of the printed document is correct. NO PROMPT is clear. AUTO ERA does not matter because SEND NOW is set so the user doesn't get a chance to press a key.

6. Editing Functions

The editor provides many editing and word processing features. These are carried out in response to the user typing an appropriate key on the keyboard, or by the same code being written from the applications program. The following sections describe each of the editing functions in some detail.

Some of the editing functions such as paragraph movement and reforming paragraphs, require fairly complex internal operations to be carried out. This often results in rather strange behaviour of the screen display, involving scrolling operations which might not be expected.

The four joystick directions and the INS, DEL and ERASE keys each have three different functions which are explained below. These functions are obtained by using the key alone, or with either the SHIFT or CTRL keys. In fact the CTRL function can also be obtained by using the ALT key.

6.1 Cursor Movement (The Joystick)

Cursor movement is the most fundamental operation for a full screen editor. On the Enterprise, cursor movement is carried out with the joystick in conjunction with the SHIFT and CTRL keys. The autorepeat on the joystick allows continuous cursor movement by just holding the joystick in one of its eight possible positions.

The cursor can be moved anywhere on the video page but cannot be moved off the page. If an attempt is made to move it off the top or bottom of the page then the display will scroll to bring more text from the buffer onto the page. This scrolling will stop when the start or end of the text is reached.

The cursor can be moved beyond the end of lines or outside the margin settings without the text being affected. However when a character is typed, extra spaces will be put in to fill up to the cursor position and word wrap may occur if the cursor is outside the margins.

Although only the four orthogonal directions of cursor movement are provided by the editor, diagonal movement is still possible. This is because if the joystick is moved to one of the diagonal positions, the keyboard driver autorepeat will return the appropriate two joystick codes alternately so the editor will execute them alternately. This is only useful for simple cursor movements, not for the shifted and controlled movements.

The possible cursor movements and their codes are:

Joystick Movement	Key Code	Function
UP	0B0h	Cursor up line
Shift-UP	0B1h	Cursor up page
Ctrl-UP	0B2h	Cursor up paragraph
DOWN	0B4h	Cursor down line
Shift-DOWN	0B5h	Cursor down page
Ctrl-DOWN	0B6h	Cursor down paragraph
LEFT	0B8h	Cursor left character
Shift-LEFT	0B9h	Cursor to start of line
Ctrl-LEFT	0BAh	Cursor left word
RIGHT	0BCh	Cursor right character
Shift-RIGHT	0BDh	Cursor to end of line
Ctrl-RIGHT	0BEh	Cursor right word

6.1.1 Left and Right by Character

The simple left and right movements move the cursor by one character left and right. There is no wrap around from one line to the next so if the cursor is at the extreme left or right of the video page then attempting to move it further will have no effect.

6.1.2 Start and End of Line

Moving the joystick left or right with the SHIFT key held down will move the cursor to the start or end of the current line respectively. In this context the start of the line is the first actual character in the line, discounting any spaces up to the left margin of that line. The end of the line is the last actual character in the line discounting any trailing spaces which are not represented in the buffer (although there may be spaces which are in the buffer).

6.1.3 Left and Right by Word

Joystick left or right with the CTRL key held down moves the cursor left or right by a word. The exact definition of a word is rather complex but is basically a string of alphanumeric characters and a string of non-alphanumeric characters in either order. Moving left or right by a word does wrap around between lines.

6.1.4 Up and Down by Line

The straightforward joystick up or down operation moves the cursor up or down by one line. The cursor always remains in the same column position. If the cursor is on the top or bottom line of the video page then, assuming that there are more lines in the buffer, the video page will be scrolled appropriately to bring another line onto the display. If the cursor is on the first or last line of the text then nothing will happen.

6.1.5 Up and Down by Page

Moving the joystick up with the SHIFT key held down will move the cursor up by a page. If the cursor is on the top line of the video page then the page will be scrolled one less lines than the height of the page so that the old top line is now the bottom line and the rest of the video page is new lines from the buffer. If there are insufficient lines in the buffer then the scrolling will stop when the first line of text is on the top line of the video page. If the cursor was not at the top of the page then it is moved to the top line of the page.

For moving down by a page the situation is analogous, with the cursor being moved to the bottom line of the page unless it is there already in wich case the page will be scrolled up. In either case the cursor will be left on the same column as it started on.

6.1.6 Up and Down by Paragraph

Moving up and down by paragraph (by using CTRL) is rather different from other up and down movements. In moving up by a paragraph the cursor will be put on the first character of the current paragraph, unless it is already on the first character in which case it will be put on the first character of the previous paragraph. The video page will of course be scrolled appropriately to ensure that the cursor remains on the screen.

Moving down by a paragraph always moves the cursor to the start of the next paragraph unless it is already in the last paragraph in which case it will be left at the end of that paragraph.

6.2 Inserting and Insert Mode Control (The INS key)

The insert key has three separate functions which are:

INS	0A8h	-	Insert a space
Shift-INS	0A9h	-	Insert a new line
Ctrl-INS	0AAh	-	Toggle insert/overwrite mode

6.2.1 Inserting Spaces and Lines

When the INS key is used on its own it simply inserts a space character before the cursor position and leaves the cursor on this space. It is useful for inserting a few characters while in overwrite mode (see below). This is done simply by inserting the correct number of spaces and then over-typing them with the required characters.

When the INS key is used with the SHIFT key, the current line will be split with a hard carriage return (end of paragraph marker) at the cursor position. If the cursor is at the start or end of a line then this will have the effect of inserting a blank line.

6.2.2 Toggle Insert and Overwrite Mode

When the INS key is used with the CTRL key, it toggles between insert and overwrite mode, the initial default being overwrite mode. The current mode is indicated by the cursor which is changed to a different character depending on the mode. For overwrite mode it is character number 14 (a rectangular block) and for insert mode it is character number 30 (a left pointing arrow).

In overwrite mode if a character is typed when there is already a character at the cursor position then the old character will be replaced by the new one. In insert mode the new character will be inserted before the old character and the old character along with the rest of the line will be moved one character to the right.

Whether overwrite or insert mode is selected also affects some details to do with word wrapping and splitting lines in the middle of the buffer.

6.3 Deleting and Erasing (The DEL and ERASE keys)

The DEL and ERASE keys have very similar functions, both delete characters from the buffer. Basically the DEL key goes rightwards while the ERASE key goes leftwards. Each key has three functions which correspond to the three types of horizontal cursor movement:

DEL	0A0h	Delete character right
Shift-DEL	0A1h	Delete line right
Ctrl-DEL	0A2h	Delete word right
ERASE	0A4h	Erase character left
Shift-ERASE	0A5h	Erase line left
Ctrl-ERASE	0A6h	Erase word left

6.3.1 Deleting And Erasing Characters

If used without the SHIFT or CTRL keys then DEL and ERASE each delete a single character and move the rest of the line left to fill up the gap. DEL deletes the character under the cursor leaving the cursor in the same position, while ERASE deletes the character to the left of the cursor and then moves the cursor onto the previous character.

Both functions wrap around between lines and thus can be used to join lines together. If the cursor is on the first character of a line then ERASE will join this line to the previous line. 'The line separator counts as one character as far as deletion goes. DEL will join the current line to the next one if it is at the end of a line.

6.3.2 Deleting and Erasing Lines

When used with SHIFT, the ERASE and DEL keys delete to the start and end of the current line respectively. If already at the start or end of the line then nothing will be done, these functions do not join lines together.

6.3.3 Deleting and Erasing Words

When DEL and ERASE are used with the CTRL key they delete one word, using the same definition of a word as cursor movement. The deletion is done by repeatedly deleting characters until the end of the word is reached. These functions will join lines together and for this purpose the line separator counts as a word. As usual DEL deletes rightwards and ERASE deletes leftwards.

6.4 The TAB key

The TAB key (key code 09h) moves the cursor to the next tab stop, or to the start of the next line if there are no more tab stops on this line. The current TAB settings can be seen on the ruler line if it is displayed.

In overwrite mode the cursor is simply moved to the next tab stop. In insert mode the next tab stop is reached by inserting spaces and moving any more text on the line to the right. When moving to the start of a new line in insert mode, spaces will be inserted up to the right margin and then a new line will be inserted (not an end of paragraph marker).

6.5 The Editing Function Keys

The more complex editing functions, and particularly the word processor type functions are carried out by using the eight function keys in conjunction with CTRL or ALT. This gives a possible 16 editing functions although only 14 of these are utilised because function key 8 is not used.

These 14 editing functions are listed here, along with their key codes and each one is then described in more detail in the following sections.

Ctrl-F1	0F0h	-	Reform paragraph
Ctrl-F2	0F1h	-	Centre line
Ctrl-F3	0F2h	-	Toggle tab
Ctrl-F4	0F3h	-	Set left margin
Ctrl-F5	0F4h	-	Release margins
Ctrl-F6	0F5h	-	Move paragraph up
Ctrl-F7	0F6h	-	Change line colour
Alt-F1	0F8h	-	Justify paragraph
Alt-F2	0F9h	-	Remove all tab stops
Alt-F3	0FAh	-	Toggle ruler line display
Alt-F4	0FBh	-	Set right margin
Alt-F5	0FCh	-	Reset margins and tabs
Alt-F6	0FDh	-	Move paragraph down
Alt-F7	0FEh	-	Change paragraph colour

6.5.1 Reform and Justify Paragraph (Ctrl-F1 and Alt-F1)

Reform and justify paragraph are very similar functions, in fact justify does exactly the same as reform but also justifies. Both operate on the paragraph containing the cursor, and leave the cursor on the start of the next paragraph. This means that pressing one of these keys repeatedly will reform (or justify) each paragraph of a document in turn, without need for using the joystick.

Reform paragraph moves to the start of the paragraph and then walks through the paragraph to the end. As it goes it adjusts the left margin of each line to be equal to the current left margin, removes any soft spaces (left over from previous justification) and word wraps each line to the current right margin, joining lines together where possible.

The result is that the new paragraph appears exactly as it would if all the characters of the paragraph were newly typed in, so any untidy sections resulting from other editing operations will be reformed.

Justify does exactly the same as reform but it also inserts soft spaces into each line of the paragraph except the last one, to ensure that each line finishes exactly on the right margin.

6.5.2 Centre Line (Ctrl-F2)

Centre line is a fairly simple function which operates on a single line, not a whole paragraph. It inserts sufficient spaces before the line to centre it between the current margins. Leading and trailing spaces are first removed to ensure that they are not included in the centring. The cursor is left on the start of the line. If the line is too long to fit between the margins then it will be left so that it starts at the left margin position.

6.5.3 Toggle Ruler Line Display (Alt-F3)

The ruler line display is a red line which can be displayed on the very top line of the video page. It indicates where the left and right margins are set and also the positions of any tab stops.

The left margin is indicated by an "L" and the right margin by an "R". Tab stops are marked by a vertical bar and other character positions between the margins are indicated by dashes. Also if the margins are released (see below) then an asterisk will be displayed in the extreme right hand position of the ruler line.

This function key simply toggles the ruler line display on and off, the default being off. All the facilities of tab stops and margins can be used regardless of whether the ruler line is displayed or not, but it can get confusing if it is not. The built in word processor sets the ruler line display on for the user.

6.5.4 Toggle and Clear Tabs (Ctrl-F3 and Alt-F2)

The toggle tab function sets a tab stop at the current cursor column, or removes it if one was already set. Tab stops can only be set between the current margin positions, although when the margins are moved tab stops which are outside them are remembered and restored when the margins are moved back out. It is advisable to have the ruler line displayed when using this function.

All tab stops can be removed by a single function key press, including those which are outside the current margin settings (and thus not visible on the ruler line). This is useful to get rid of the standard tabs before setting up your own set.

When an editor channel is opened, the tab stops are set up by default to every eight character positions since this corresponds to the standard setting for tabs on machines with fixed tab stops.

6.5.5 Set Left and Right Margins (Ctrl-F4 and Alt-F4)

The left and right margins define what portion of the video page is used for entering and displaying text. When an editor channel is opened the margins are initialised to the widest possible setting. The user can set new margin postions by putting the cursor on the desired column and pressing the appropriate function key.

The right margin can be in any column up to two less than the video page width. Thus for a 40 column display (BASIC's default) the right margin can be any column up to 38. The left margin can be in any column from 1 up to one less than the right margin column. The default margin settings for BASIC's default channel are thus: left margin at column 1, right margin at column 38.

An attempt to set an illegal margin position will result in both margins being reset to their default settings. The applications program can use these codes to set margin positions but there is also a special function call which can be used to set the margins and also to read their current settings. This is described later.

6.5.6 Release Margins (Ctrl-F5)

As mentioned before there is a margin release function. This is in fact a toggle action, it releases margins on the first press and the re-enables them when it is pressed again. When margins are released the margins remain displayed on the ruler line and an asterisk is displayed on the extreme right hand end.

When margins are released, all operations which normally use the margin settings use the default settings instead. Thus word wrapping will occur at the last-but-two column rather than the right margin and characters may be typed in outside the margins.

6.5.7 Reset Margins and Tabs (Alt-F5)

This function key resets the margin settings to their default values and sets up the default positions of tab stops (every eight columns).

6.5.8 Move Paragraph Up and Down (Ctrl-F6 and Alt-F6)

These functions can be used to move a paragraph up or down. Each key press will move the paragraph up or down by one line, unless it is already at the start or end of the buffer. To move a paragraph by more than one line this key should be pressed repeatedly until the paragraph reaches the desired position.

The paragraph to be moved is defined to start on the current cursor line, and end at the next end of paragraph marker. Thus to move a complete paragraph the cursor should first be positioned on the first line of the paragraph. This definition of a paragraph has to be used to ensure that one paragraph can be moved through another correctly.

Although not essential it is useful to use the "colour paragraph" function before doing a series of moves up or down. This highlights the paragraph to make it easy to see what is going on, and also puts the cursor at the start so that the whole paragraph will be moved.

6.5.9 Colour Line and Paragraph (Ctrl-F7 and Alt-F7)

These functions can be used to change the colour of the text. The colour line function just affects the current line whereas the colour paragraph function affects the entire current paragraph, and has the side effect of moving the cursor to the start of the paragraph ready for paragraph moving.

Each change colour operation changes the colour to the next one of four posible colour pairs, cycling back to the first pair after the fourth. A colour pair specifies which paper and ink palette colour the the video driver will use for the text. The four colour pairs are (in order): (0,1), (2,3), (4,5), (6,7), with (0,1) being the default (normally green on black).

Note that if the editor is using a hardware text page then colour pairs (4,5) and (6,7) will in fact appear as pairs (0,1) and (2,3) since the video driver only supports two colour pairs in this mode.

7. Special Function Calls

7.1 Setting Margin Positions

The margin positions can be set by the user or the applications program by using a function key. However a more general way for the applications program to set the margin positions is provided as a special function call. The parameters for this call are:

Parameters:
A = Channel number (1...255)
B = @@MARG (=24) (Special function code)
D = New right margin column
E = New left margin column

Returns:
A = Status
D = Right margin column
E = Left margin column

The column numbers can be from 1 up to the width of the video page, with the usual restrictions on valid margin settings. There are two special values which can be given for the new left and right margin parameters. If either of them is 0FFh then the current cursor column will be used for that margin. If either of them is zero then that margin setting will be unchanged. Thus the current margin settings can be read without affecting them, simply by setting both D and E to zero.

7.2 Loading and Saving Document Files

Two special function calls are provided for loading and saving document files. The format of saved documents fits in with the standard EXOS file module format which is described in the EXOS kernel specification. The module type for editor document files is $$EDIT (=8) and the module header contains no other information.

The simplest of the two functions is saving. The call must specify a channel number down which the document is to be saved and this channel must be opened before making the call. The editor will write a suitable module header followed by the data of the document. It does not write out an end of file header at the end, since the application program may want to create a multi-module file.

The details of this special function call are:

Parameters: A = Editor channel number (1...255)
B = Channel number to write to (1...255)

Returns: A = Status

Loading is slightly more complex. Before making the special function call, the file to load from must have been opened, and a file header with type $$EDIT must have been read in. The editor is then called, giving the channel number to load from. It first clears all the text out of the editor buffer and then loads the new document in from the file. It loads line by line and checks that each line is valid before going on to the next. If it finds an invalid line or the buffer becomes full then it stops loading and returns an error (.EDINV or .EDBUF). All previous lines in the buffer will be valid. The cursor will be left at the start of the document.

The parameters for the loading special function call are:

Parameters: A = Editor channel number (1...255)
B = Channel number to load from (1...255)

Returns: A = Status

A saved document contains the characters of the text and also information about paragraph structure, the left margin position for each line, and the colour of each line. Thus if a document is saved and loaded into an editor channel with the same sized buffer and video page, then it will re-load exactly as it was before being saved.

Information about editor options and so on is not saved with the file so any tab settings, current margin settings and so on will have to be set up again when the file is loaded.

A document can be loaded into a different sized editor channel but may have to have some lines adjusted if their left margin setting is invalid. Also it may not all fit into the buffer if the new buffer is smaller.

8. Error Handling

The only error which can be generated inside the editor during normal operation (apart from loading and saving documents, opening channels, illegal special function calls, and unknown escape sequences) is .VCURS which is returned if an invalid cursor position is given in the cursor positioning escape sequence.

However, since the editor is communicating with a video page and a keyboard channel, it can get errors from these channels. Generally an error from one of these channels means that the channel is misbehaving in some way. For example if someone has re-directed the editor's video channel to a different device, or even closed it then the editor will get errors from its video channel.

If an error does come from one of these channels then the editor returns either .EKEY or .EVID error code to the applications program. It also remembers that this error occurred and whenever it is next called, with any EXOS call, it checks the channel (by writing a null to the video or reading status from the keyboard) to see if it is basically working. If it is working then it clears the error flag and carries on normally. If it is not working then it returns the appropriate error code (.EKEY or .EVID) to the user without carrying out the function call.

This procedure ensures that the applications program can tell when the editor is having trouble with its secondary channels and attempt to put things right. This is how BASIC manages to recover if the editor's video channel is closed. BASIC discovers this and re-opens it again.

Another feature is that if a NULL (ASCII code 0) is written to the editor, instead of just being ignored, it is written directly to the video page, which will ignore it. This provides a way to 'poke' the editor to check if its video channel is still there, without having any effect on the editor's data.

9. Quick Reference Summary

9.1 EXOS Calls

OPEN/CREATE CHANNEL - Treated identically. Supports multiple channels. Device name "EDITOR:". Filename and unit number ignored. EXOS variables VID_EDIT, KEY_EDIT, BUF_EDIT must be set before open.

CLOSE/DESTROY CHANNEL - Treated identically.

READ CHARACTER/BLOCK - Returns characters from buffer after allowing user to do editing. Details controlled by FLG_EDIT EXOS variable.

WRITE CHARACTER/BLOCK - Printing characters put in buffer and displayed. Responds to some control codes and ESC=. Some codes above 0A0h do editing functions.

READ STATUS - Returns C=0 if a read character call would return character immediately without allowing user a chance to edit. Returns C=1 otherwise, or C=0FFh if just finished a SEND ALL.

SET STATUS - Not supported.

SPECIAL FUNCTION - @@MARG = 24 Set and read margins.
@@CHLD = 25 Load document file.
@@CHSV = 26 Save document file.

9.2 EXOS Variables

VID_EDIT - Channel number of video page.
KEY_EDIT - Channel number of keyboard channel.
BUF_EDIT - Buffer size in multiples of 256 bytes.

FLG_EDIT - Flags to control reading from editor.
b7 - SEND NOW
b6 - SEND ALL
b5 - NO READ
b4 - NO SOFT
b3 - NO PROMPT
b2 - AUTO ERA
b0 & b1 - Not used.

++++++++++ END OF DOCUMENT ++++++++++