0. INTRODUCTION

This is not a formal specification of the editor. It is just an outline of the editor's functions, concentrating on the interface between the editor and an applications program (eg. Lisp or BASIC). It is intended to contain sufficient information to allow an applications program to be fully interfaced to the editor. It should be read in conjunction with the specifications of the EXOS kernel (ET10/8), video driver (ET11/8) and keyboard driver (ET13/4).

1. Outline of Editor Operation

The editor is an EXOS device which the applications program can read and write to. It can support any number of channels open to it at a time, each channel corresponds to a seperate "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 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 useful as the keyboard driver only allows one channel to be open to it), but must have seperate 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 channel in such a way that it is kept updated to form a "window" onto the text buffer, although the video channel has its own copy of the text it is displaying.

2. Opening Channels

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 must be opened before opening the editor channel. The video page must be a text mode and must be at least 2 rows by 4 characters. The editor determines the size of the video page in its open channel routine and complains if it is too small. Note that the editor does not display the video page on the screen. It is up to the applications program to do a special function call to the video page to display it if this is what is wanted (see ET11/8).

The actual size of the editor's text buffer 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. 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. Writing to the Editor

Any printing characters written to the editor will be put into the text buffer at the current cursor position and displayed on the video page. Codes in the range 080h to 09Fh are also interpreted as printing characters and displayed on the video page. The video driver will display these as characters 00h to 01Fh from the font, which are normally un-obtainable because 00h to 01Fh are interpreted as control codes.

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 (see ET11/8).

Codes above 0A0h are interpreted exactly as if they had been received from the keyboard. These provide various editing functions and cursor movement, but are really provided for demonstration purposes. See below for the codes and their functions.

4. 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 afterwards.

The assignment of bits in this byte 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
```

4.1 Basic Editor Read Action

With all editor flags zero, when the editor receives a read character call, 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.

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 and carry out various editing functions. The details of this are given below in the section on editing. 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.

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 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.

Note that the editor flags are sampled once when the first read character call is made and then not again untill all the text has been sent. Changing them in the meantime will therefore have no effect on the read which is in operation. If a write character call is mode while a read is in progress then th einternal flag is cleared so that read will be aborted.

4.2 The Editor Read Flags in Detail

4.2.1 The SEND NOW Flag (bit-7)

This flag is set to do an immediate read from the editor, rather than waiting for the user to press ENTER. The text which is returned is as defined by the other flags.

4.2.2 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!). 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 stop the applications program reading. 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.

4.2.3 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.

4.2.4 The NO SOFT Flag (bit-4)

This flag controls the sending of soft carriage returns and soft spaces. Soft carriage returns are those which seperate successive lines of a single paragraph (continuation lines if you like). 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.

4.2.5 The NO PROMPT Flag (bit-3)

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

This 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.

2.2.6 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 specially 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.

2.2.7 Typical Flag Combinations.

The use of these flags can be rather confusing so some typical combinations are given here. It is left as an exercise for the reader to understand why these combinations are used for the various purposes. (x indicates don't care.)

```
000101xx  -  BASIC reading a command.
000110xx  -  BASIC doing an INPUT command.
001xx0xx  -  WORD PROCESSOR in editing mode.
11000xxx  -  WORD PROCESSOR reading a document for
             printing.

????????  -  LISP ?
```


3. Editing Functions

This is a list of all the keyboard editing functions together with the codes which trigger them. These codes can also be sent for the applications program.

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)	-	Colour line
Ctrl-F8	(0F7h)	-	not used
Alt-F1	(0F8h)	-	Justify paragraph
Alt-F2	(0F9h)	-	Clear tabs
Alt-F3	(0FAh)	-	Toggle ruler
Alt-F4	(0FBh)	-	Right margin
Alt-F5	(0FCh)	-	Reset margins and tabs
Alt-F6	(0FDh)	-	Move paragraph down
Alt-F7	(0FEh)	-	Colour paragraph
Alt-F8	(0FFh)	-	not used
INS	(0A8h)	-	Insert a space
Shift-INS	(0A9h)	-	Insert a new line.
Ctrl-INS	(0AAh)	-	Toggle insert/overwrite mode.
DEL	(0A0h)	-	Delete character right
Shift-DEL	(0A1h)	-	Delete line right
Ctrl-DEL	(0A2h)	-	Delete word right
ERA	(0A4h)	-	Erase character left
Shift-ERA	(0A5h)	-	Erase line left
Ctrl-ERA	(0A6h)	-	Erase word left
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

