Date - July 14, 1986

Author - Cheryl Ewy

Subject - LineEdit ERS

Document Version Number - 00:10

Revision History

00:00	(06-17-86)	Initial Release
00:10	(07-14-86)	LEFromScrap routine added LEToScrap routine added Call numbers changed for some routines Support for control-F, control-Y and control-X added to LEKey Support for triple-click added to LEClick LETextBox information updated Error codes updated

OVERVIEW

LineEdit is a set of routines that provide basic line editing capabilities. These capabilities include:

- Inserting new text.
- Deleting characters that are backspaced over.
- Translating mouse activity or arrow keys into text selection.
- Deleting selected text and possibly inserting it elsewhere, or copying text without deleting it.

The LineEdit routines follow the Apple Human Interface Guidelines and support these standard features:

- Positioning the insertion point by clicking the mouse.
- Moving the insertion point 1 character at a time by using the left and right arrow keys.
- Moving the insertion point 1 word at time by using Option-LeftArrow or Option-RightArrow.
- Moving the insertion point to the beginning or end of the line by using OpenApple-LeftArrow or OpenApple-RightArrow.
- Selecting text by clicking and dragging with the mouse.
- Selecting text by using Shift-LeftArrow and Shift-RightArrow.
- Selecting words by double-clicking the mouse.
- Selecting words by using Shift-Option-LeftArrow or Shift-Option-RightArrow.
- Selecting the whole line by triple-clicking the mouse.
- Selecting from the insertion point to the beginning or end of the line by using Shift-OpenApple-LeftArrow or Shift-OpenApple-RightArrow.
- Extending or shortening the selection by clicking the mouse while holding down the Shift key.
- Deleting the selection or the character to the left of the insertion point by using Backspace.
- Deleting the selection or the character to the right of the insertion point by using Control-F.
- Deleting the selection or the whole line by using Control-X.
- Deleting the selection or from the insertion point to the end of the line using Control-Y.
- Inverse highlighting of the current text selection, or display of a blinking vertical bar at the insertion point.

- Cutting (or copying) and pasting. LineEdit puts text you cut or copy into the LineEdit scrap.

LineEdit does not support:

- more than 256 characters per line (except when using LETextBox)
- line wrap (except when using LETextBox)
- centered or right-justified text (except when using LETextBox)
- fully justified text (text aligned with both the left and right margins)
- scrolling
- the use of more than one font or stylistic variation per line
- "intelligent" cut and paste (adjusting spaces between words during cutting and pasting)
- tabs

EDIT RECORDS

To edit a line of text on the screen, LineEdit needs to know where and how to display the text, where to store the text, and other information related to editing. This display, storage, and editing information is contained in an **edit record** that defines the complete editing environment.

You prepare to edit text by specifying a destination rectangle in which to draw the text and a view rectangle in which the text will be visible. LineEdit incorporates the rectangles and the drawing environment of the current grafPort into an edit record, and returns a handle to the record. Most of the LineEdit routines require you to pass this handle as a parameter.

In addition to the two rectangles and a description of the drawing environment, the edit record also contains:

- a handle to the text to be edited
- a pointer to the grafPort in which the text is displayed
- the current selection range, which determines which characters will be affected by the next editing operation

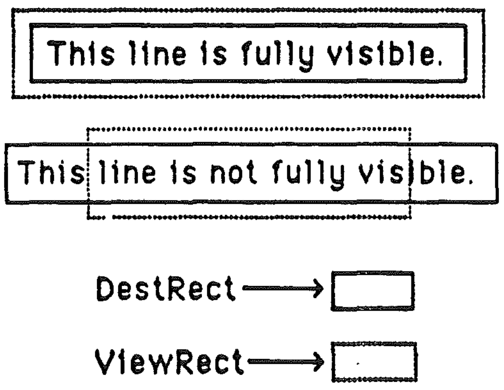
For most operations, you don't need to know the exact structure of an edit record since the LineEdit routines access the record for you. The structure of an edit record is given below.

TextHandle	HANDLE	{hndl to text to be edited}
Length	INTEGER	{current length of text}
MaxLength	INTEGER	{maximum text length}
DestRect	Rect	{destination rectangle}
ViewRect	Rect	{view rectangle}
PortPtr	POINTER	{ptr to grafPort}
LineHite	INTEGER	{used for highlighting}
BaseHite	INTEGER	{used for drawing the text}
SelStart	INTEGER	{start of selection range}
SelEnd	INTEGER	{end of selection range}
ActFlg	INTEGER	{used internally}
CarAct	INTEGER	{used internally}
CarOn	INTEGER	{used internally}
CarTime	LONGINT	{used internally}
HiliteHook	POINTER	{ptr to highlight routine}
CaretHook	POINTER	{ptr to caret routine}

Warning: Never change any of the fields in the edit record directly. The fields can only be changed by calling LineEdit routines.

The DestRect and ViewRect Fields

The destination rectangle is the rectangle in which the text is drawn. The view rectangle is the rectangle within which the text is actually visible. In other words, the view of the text drawn in the destination rectangle is clipped to the view rectangle. The view rectangle also determines the area in which mouse activity affects the text. Clicking or dragging the mouse outside the view rectangle does not affect the insertion point or selection range. In most cases, the view rectangle should be a few pixels larger than the destination rectangle on all sides. This provides some slop area around the text in which the mouse activity will still have an effect on the text.

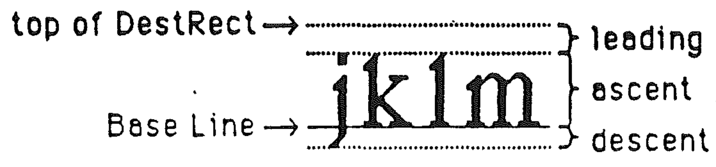


You specify both rectangles in the local coordinates of the grafPort. To ensure that the first and last characters in each line are legible in a document window, you may want to inset the destination rectangle at least four pixels from the left and right edges of the grafPort's portRect.

Edit operations may of course lengthen or shorten the text. If the text becomes too long to be enclosed by the destination rectangle, it's simply drawn beyond the right edge. LineEdit doesn't support scrolling or wrapping to the next line.

The LineHite and BaseHite Fields

The BaseHite field has to do with where the the text is drawn relative to the top of the DestRect. The LineHite field has to do with where the caret or highlighting of the selection range is drawn relative to the text. The BaseHite field specifies the distance between the top of the DestRect and the base line (leading + ascent). The LineHite field specifies the height of the line (leading + ascent + descent).

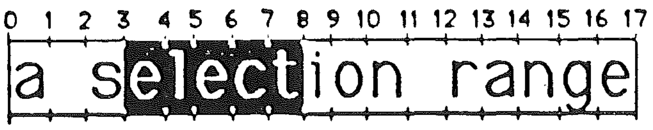


The SelStart and SelEnd Fields

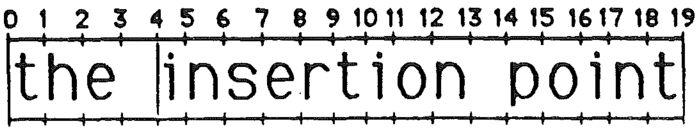
In the text editing environment, a **character position** is an index into the text, with position 0 corresponding to the first character. The edit record includes fields for character positions that specify the beginning and end of the current selection range, which is the series of characters where the next editing operation will occur. For example, the procedures that cut or copy from the text of an edit record do so to the current selection range.

The selection range, which is inversely highlighted when the window is active, extends from the beginning character position to the end character position. The figure below shows a selection range between positions 3 and 8, consisting of five characters (the character at position 8 isn't included). The end position of a selection range may be 1 greater than the position of the last character of the text, so that the selection range can include the last character.

If the selection range is empty - that is, its beginning and end positions are the same - that position is the text's insertion point, the position where characters will be inserted. By default, it's marked with a blinking caret (actually a vertical bar).



selection range
beginning at position 3
and ending at position 8



insertion point
at position 4

If you call a procedure to insert characters when there's a selection range of one or more characters rather than an insertion point, the editing procedure automatically deletes the selection range and replaces it with an insertion point before inserting the characters.

The HiliteHook and CaretHook Fields

The HiliteHook and CaretHook fields are used for text highlighting and drawing the caret. These fields are initialized to $00000000. You can set the contents of these fields by calling the LESetHilite and LESetCaret routines.

If you store the address of a routine in HiliteHook, that routine will be used instead of the QuickDraw procedure InvertRect whenever a selection range is to be highlighted. For example, you can write a routine which underlines selection ranges instead of highlighting them. The routine will be called with the stack containing a pointer to the rectangle enclosing the text being highlighted.

The routine whose address is stored in CaretHook acts exactly the same way as the HiliteHook routine, but on the caret instead of the selection highlighting, allowing you to change the appearance of the caret. The routine will be called with the stack containing a pointer to the rectangle that encloses the caret.

USING LINE EDIT

Before using LineEdit, you must initialize the Memory Manager, QuickDraw, the Event Manager and the Window Manager, in that order.

The first LineEdit routine to call is the initialization routine LEStartUp.

Call LENew to allocate an edit record; it returns a handle to the record. Most of the text editing routines require you to pass this handle as a parameter.

When you're completely done with an edit record and want to dispose of it, call LEDispose.

To make a blinking caret appear at the insertion point, call the LEIdle routine as often as possible (at least once each time through the main event loop); if it's not called often enough, the caret will blink irregularly.

When a mouse-down event occurs in the view rectangle (and the window is active) call the LEClick routine. LEClick controls the placement and highlighting of the selection range in response to mouse activity, including supporting use of Shift-Click to make extended selections.

Key-down, auto-key, and mouse events that pertain to text editing can be handled by several LineEdit routines:

- LEKey inserts characters, deletes characters backspaced over, controls the placement and highlighting of the selection range in response to the LeftArrow and RightArrow keys, and handles the Control-F, Control-X and Control-Y commands.

- LECut transfers the selection range to the LineEdit scrap, removing the selection range from the text.

- LEPaste inserts the contents of the LineEdit scrap. By calling LECut, changing the insertion point, and then calling LEPaste, you can perform a "cut and paste" operation, moving text from one place to another.

- LECopy copies the selection range to the LineEdit scrap. By calling LECopy, changing the insertion point, and then calling LEPaste, you can make multiple copies of text.

- LEDelete removes the selection range (without transferring it to the scrap). You can use LEDelete to implement the Clear command.

- LEInsert inserts specified text. Since LEDelete and LEInsert do not modify the scrap, they're useful for implementing the Undo command.

After each editing procedure, LineEdit redraws the text if necessary from the insertion point to the end of the text. You never have to set the selection range or insertion point yourself; LEClick and the editing routines leave it where it should be. If you want to modify the selection range directly, however - to highlight an initial default name or value, for example - you can use the LESetSelect routine.

To implement cutting and pasting of text between different applications, or between applications and desk accessories, you need to transfer the text between the LineEdit scrap (which is a private scrap used only by LineEdit) and the Scrap Manager's desk scrap. To do this, use the LEFromScrap and LEToScrap routines.

When an update event is reported for a text editing window, call LEUpdate (along with the Window Manager routines BeginUpdate, EraseRect and EndUpdate) to redraw the text.

> **Note:** After changing any fields of the edit record that affect the appearance of the text, you should call the Window Manager routine InvalRect so that the text will be updated.

The LEActivate and LEDeactivate routines must be called each time GetNextEvent reports an activate event for a text editing window. LEActivate simply highlights the selection range or displays a caret at the insertion point; LEDeactivate unhighlights the selection range or removes the caret.

The LESetText routine lets you change the text being edited. For example, if your application has several separate pieces of text that must be edited one at a time, you don't have to allocate an edit record for each of them. Allocate a single edit record, and then use LESetText to change the text.

If you want to draw noneditable text in any given rectangle, you can use the LETextBox routine.

LINE EDIT ROUTINES

HouseKeeping

```
LEBootInit             Call # $01
```

LEBootInit is called at boot time. It does nothing.

```
LEStartUp              Call # $02

       input           ZeroPageAdrs     INTEGER
       input           ProgramID        INTEGER
```

LEStartUp Initializes LineEdit and allocates a handle for the LineEdit scrap. The scrap is initially empty. ZeroPageAdrs is the starting address in Bank 0 of a 1-page work area assigned to LineEdit. ProgramID is the ID LineEdit will use when getting memory from the Memory Manager. Duplicate LEStartUp calls will cause an error to be returned.

Note: You should call LEStartUp even if your application doesn't use LineEdit, so that desk accessories and dialog and alert boxes will work correctly.

```
LEShutDown              Call # $03
```

LEShutDown shuts down LineEdit and releases any workspace allocated to it.

```
LEVersion               Call # $04

     input              Result space    WORD
     output             VersionInfo     INTEGER
```

LEVersion returns identifying information for LineEdit.

```
LEReset                 Call # $05
```

LEReset returns an error if LineEdit is active, otherwise it does nothing.

```
LEActive                Call # $06

     input              Result space    WORD
     output             ActiveFlag      INTEGER
```

LEActive returns a non-zero value if LineEdit is active, otherwise it returns a 0.

Edit Record Allocation

```
LENew                   Call # $09

     input              Result space    LONG WORD
     input              DestRectPtr     POINTER to Rect
     input              ViewRectPtr     POINTER to Rect
     input              MaxTextLen      INTEGER
     output             hLE             HANDLE
```

LENew allocates space for the text, creates and initializes an edit record, and returns a handle to the new edit record. DestRect and ViewRect are the destination and view rectangles, respectively. Both rectangles are specified in the current grafPort's coordinates. The view rectangle must not be empty. For example, don't make its right edge less than its left edge. If you don't want any text visible - specify a rectangle off the screen instead. MaxTextLen specifies how many bytes to allocate for the text. It should be a value from 1 to 256. The text will be limited to this length.

Call LENew once for every edit record you want allocated. The edit record incorporates the drawing environment of the grafPort, and is initialized with an insertion point at character position 0.

Note: The caret won't appear until you call LEActivate.

```
LEDispose               Call # $0A

      input             hLE                 HANDLE
```

LEDispose releases the memory allocated for the edit record and text specified by hLE. Call this procedure when you're completely through with an edit record.

Changing the Text of an Edit Record

```
LESetText               Call # $0B

      input             TextPtr             POINTER
      input             Length              INTEGER
      input             hLE                 HANDLE
```

LESetText incorporates a copy of the specified text into the edit record specified by hLE. The TextPtr parameter points to the text, and the Length parameter indicates the number of characters in the text. If Length is greater than the maximum text length allowed for the edit record, only the maximum number of characters allowed will be copied into the edit record. The selection range is set to an insertion point at the end of the text. LESetText doesn't affect the text currently drawn in the destination rectangle, so call InvalRect afterward if necessary.

Insertion Point and Selection Range

```
LEIdle                  Call # $0C

      input             hLE                 HANDLE
```

LEIdle should be called repeatedly to make a blinking caret appear at the insertion point (if any) in the text specified by hLE. (The caret appears only when the window containing that text is active.) LineEdit observes a minimum blink interval: No matter how often LEIdle is called, the time between blinks will never be less than the minimum interval. The user can adjust the minimum blink interval with the Control Panel desk accessory.

To provide a constant frequency of blinking, LEIdle should be called as often as possible - at least once each time through the main event loop. Call it more than once if your application does an unusually large amount of processing each time through the loop.

> **Note:** LEIdle actually only needs to be called when the window containing the text is active.

```
LEClick              Call # $0D

     input           EventPtr        POINTER to event record
     input           hLE             HANDLE
```

LEClick controls the placement and highlighting of the selection range as determined by mouse events. Call LEClick whenever a mouse-down event occurs in the view rectangle of the edit record specified by hLE, and the window associated with that edit record is active. The EventPtr parameter should be a pointer to the mouse-down event record.

LEClick unhighlights the old selection range unless the selection range is being extended. If the mouse moves, meaning that a drag is occurring, LEClick expands or shortens the selection range accordingly. In the case of a double-click, the word under the cursor becomes the selection range; dragging expands or shortens the selection a word at a time. In the case of a triple-click, the entire line becomes the selection range. LEClick keeps control until the mouse button is released.

```
LESetSelect          Call # $0E

     input           SelStart        INTEGER
     input           SelEnd          INTEGER
     input           hLE             HANDLE
```

LESetSelect sets the selection range to the text between SelStart and SelEnd in the text specified by hLE. The old selection range is unhighlighted, and the new one is highlighted. If SelStart equals SelEnd, the selection range is an insertion point, and a caret is displayed.

SelEnd and SelStart can range from 0 to 256. SelStart must be <= SelEnd. If SelEnd is anywhere beyond the last character of the text, the position just past the last character is used.

```
LEActivate           Call # $0F

     input           hLE             HANDLE
```

LEActivate highlights the selection range in the view rectangle of the edit record specified by hLE. If the selection range is an insertion point, it displays a caret there. This procedure should be called every time the Event Manager routine

GetNextEvent reports that the window containing the edit record has become active.

```
LEDeactivate          Call # $10

      input           hLE                 HANDLE
```

LEDeactivate unhighlights the selection range in the view rectangle of the edit record specified by hLE. If the selection range is an insertion point, it removes the caret. This procedure should be called every time the Event Manager routine GetNextEvent reports that the window containing the edit record has become inactive.

Editing

```
LEKey                 Call # $11

      input           Key                 WORD
      input           Modifiers           WORD
      input           hLE                 HANDLE
```

LEKey replaces the selection range in the text specified by hLE with the character given by the Key parameter, and leaves an insertion point just past the inserted character. If the selection range is an insertion point, LEKey just inserts the character there.

If the Key parameter contains a Backspace character, the selection range or the character immediately to the left of the insertion point is deleted. If the Key parameter contains a Control-F character, the selection range or the character immediately to the right of the insertion point is deleted. If the Key parameter contains a Control-X character, the selection range or the entire line is deleted. If the Key parameter contains a Control-Y character, the selection range or the text from the insertion point to the end of the line is deleted.

If the Key parameter contains a LeftArrow or RightArrow character, LEKey will move the insertion point or extend the selection range depending on the contents of the Modifiers parameter.

LEKey redraws the text as necessary.

Call LEKey every time the Event Manager routine GetNextEvent reports a keyboard event that your application decides should be handled by LineEdit. The Key parameter should be the key reported by the event record. The Modifiers parameter should be a copy of the Modifiers field in the event record.

> **Note:** LEKey inserts every character passed in the Key parameter (except for Backspace, Control-F, Control-X, Control-Y, LeftArrow

and RightArrow), so it's up to the application to filter out all characters that aren't actual text (such as Command keys and other Control characters).

```
LECut                    Call # $12

      input              hLE                 HANDLE
```

LECut removes the selection range from the text specified by hLE and places it in the LineEdit scrap. The text is redrawn as necessary. Anything previously in the scrap is deleted. If the selection range is an insertion point, the scrap is emptied.

```
LECopy                   Call # $13

      input              hLE                 HANDLE
```

LECopy copies the selection range from the text specified by hLE into the LineEdit scrap. Anything previously in the scrap is deleted. The selection range is not deleted. If the selection range is an insertion point, the scrap is emptied.

```
LEPaste                  Call # $14

      input              hLE                 HANDLE
```

LEPaste replaces the selection range in the text specified by hLE with the contents of the LineEdit scrap, and leaves an insertion point just past the inserted text. The text is redrawn as necessary. If the scrap is empty, the selection range is deleted. If the selection range is an insertion point, LEPaste just inserts the scrap there.

```
LEDelete                 Call # $15

      input              hLE                 HANDLE
```

LEDelete removes the selection range from the text specified by hLE, and redraws the text as necessary. LEDelete is the same as LECut (above) except that it doesn't transfer the selection range to the scrap. If the selection range is an insertion point, nothing happens.

```
LEInsert                 Call # $16

      input              TextPtr             POINTER
      input              Length              INTEGER
      input              hLE                 HANDLE
```

LEInsert takes the specified text and inserts it just before the selection range into the text indicated by hLE, redrawing the text as necessary. The TextPtr parameter points to the text to be inserted, and the Length parameter indicates the number of characters to be inserted. LEInsert doesn't affect either the current selection range or the scrap.

Text Display

```
LEUpdate              Call # $17

     input            hLE                HANDLE
```

LEUpdate redraws the text specified by hLE. Call LEUpdate every time the Event Manager routine GetNextEvent reports an update event for a text editing window - after you call the Window Manager routines BeginUpdate and EraseRect, and before you call the Window Manager routine EndUpdate. If you don't include the EraseRect call, the caret may sometimes remain visible when the window is deactivated.

```
LETextBox             Call # $18

     input            TextPtr            POINTER
     input            Length             INTEGER
     input            BoxPtr             POINTER to a Rect
     input            Just               INTEGER
```

LETextBox draws the specified text in the rectangle indicated by the BoxPtr parameter, with justification Just. LETextBox does an EraseRect on the rectangle before drawing the text and clips the text to the rectangle. LETextBox is not limited to a single line on the screen as the other LineEdit routines are. LETextBox will wrap to the next line whenever a CR (ASCII $0D) character occurs in the text string. The text string must end with a CR character.

The TextPtr parameter points to the text, and the Length parameter indicates the length of the text including the CR characters. The Length parameter can range from 0 to 32,767. The rectangle is specified in local coordinates. The Just parameter should be set to 0 for left justified text, 1 for centered text , and -1 for right justified text.

LETextBox creates its own edit record, which it deletes when it's finished, so the text it draws cannot be edited. LETextBox does not allocate space for the text string or make any copies of the text string.

Scrap Handling

```
LEFromScrap          Call # $19
```

LEFromScrap copies the desk scrap to the LineEdit scrap. If the number of characters in the desk scrap is > 256, an error is returned and the scrap is not copied.

```
LEToScrap            Call # $1A
```

LEToScrap copies the LineEdit scrap to the desk scrap.

```
LEScrapHandle        Call # $1B

     input           Result space    LONG WORD
     output          ScrapHndl       HANDLE
```

LEScrapHandle returns a handle to the LineEdit scrap.

```
LEGetScrapLen        Call # $1C

     input           Result space    WORD
     output          ScrapLength     INTEGER
```

LEGetScrapLen returns the size of the LineEdit scrap in bytes.

```
LESetScrapLen        Call # $1D

     input           NewLength       INTEGER
```

LESetScrapLen sets the size of the LineEdit scrap to the given number of bytes. NewLength should be a value from 0 to 256. If NewLength is > 256, it is set to 256.

Setting HiliteHook and CaretHook

```
LESetHilite          Call # $1E

     input           HiliteAdrs      POINTER
     input           hLE             HANDLE
```

LESetHilite sets the HiliteHook field to HiliteAdrs which should be the address of a routine which will be used to do highlighting of the selection range.

```
LESetCaret          Call # $1F

     input          CaretAdrs        POINTER
     input          hLE              HANDLE
```

LESetCaret sets the CaretHook field to CaretAdrs which should be the address of a routine which will be used to draw the caret.

LINE EDIT ERROR CODES

$1401	Duplicate LEStartUp call
$1402	Reset error
$1403	LineEdit not active
$1404	Desk scrap too big to copy