

## **Early Morning Editor Control Help**

**[Properties](#)**

**[Events](#)**

**[Ordering](#)**

**[Tips](#)**

**Tips...**

**[Disabling Nag Screens in Applications Built with Unregistered VBX](#)**

## **Disabling Nag Screens in Applications Built with Unregistered VBX...**

The Early Morning Editor Control remembers if a valid licence file (which you receive when you register the control) was available at design time. Later, when a control is created at run-time, if a licence file was not available at design time then a nag screen will be displayed. This is the normal course of affairs when evaluating an unregistered version of the control, however, after you register you will want to disable the nag screens in any applications that you built with the unregistered control. To do this you simply load a project that uses the control and then View each form in the project that uses the control. Viewing the forms causes each control to be loaded at design time, the controls will remember the fact that a valid licence file was available when they were loaded and no nag screens will be displayed at run-time. Be sure that the licence file EMEDIT.LIC is in the same directory as EMEDIT.VBX.

If you are working in C++ then loading your project and viewing the controls in your resource editor will have the same effect of disabling nag screens.

## Properties...

Those properties that do not have associated text are standard Visual Basic properties, for information on these properties see the Microsoft Visual Basic Language Reference.

[Action](#)  
[BackColor](#)  
[BorderStyle](#)  
[BottomMargin](#)  
[Caption](#)  
[CanUndo](#)  
[CanRedo](#)  
[CaretHeight](#)  
[CaretWidth](#)  
[CaretX](#)  
[CaretY](#)  
[Count](#)  
[DragMode](#)  
[DragIcon](#)  
[Enabled](#)  
[FileOpen](#)  
[FileSave](#)  
[FontBold](#)  
[FontItalic](#)  
[FontName](#)  
[FontSize](#)  
[FontStrike](#)  
[ForeColor](#)  
[FullLinesPerWindow](#)  
[Height](#)  
[HelpContextID](#)  
[hFont](#)  
[hWnd](#)  
[Index](#)  
[InsertMode](#)  
[IsDirty](#)  
[Left](#)  
[LeftHi](#)  
[LeftMargin](#)  
[LinesPerWindow](#)  
[LinkItem](#)  
[LinkMode](#)  
[LinkTimeOut](#)  
[LinkTopic](#)  
[MousePointer](#)  
[Name](#)  
[Parent](#)  
[ReadOnly](#)  
[Redraw](#)  
[RightHi](#)  
[RightMargin](#)  
[ScrollBars](#)  
[SearchCaseSensitive](#)  
[SearchOrigin](#)  
[SearchReplacement](#)  
[SearchResult](#)

SearchTarget  
SearchTo  
SearchWholeWordsOnly  
SelDefaultType  
SelEndX  
SelEndY  
SelForeColor  
SelMark  
SelStartX  
SelStartY  
SelText  
TabCount  
TabDefaultWidth  
TabMark  
TabIndex  
TabStop  
Tag  
Text  
TextIndex  
Top  
TopMargin  
TopY  
UndoLimit  
Visible  
Width  
WrapX  
WrapWholeWords

## Events...

Those events that do not have associated text are standard Visual Basic events, for information on these events see the *Microsoft Visual Basic Language Reference*.

[BeginMessage](#)

[CaretChange](#)

[Change](#)

[Click](#)

[DblClick](#)

[DragDrop](#)

[DragOver](#)

[EndMessage](#)

[FindWordBreak](#)

[GotFocus](#)

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

[LinkError](#)

[LinkOpen](#)

[LinkClose](#)

[LinkNotify](#)

[LinkChange](#)

[LostFocus](#)

[MouseDown](#)

[MouseMove](#)

[MouseUp](#)

[ProcessMessage](#)

[SearchReplaceConfirm](#)

[SelChange](#)

[TopYChange](#)

## Ordering...

Unlicensed copies of the Early Morning Editor Control are 100% fully functional so that you can thoroughly evaluate it. Unlicensed copies also have a nag screen that appears at most once whenever the control is loaded into memory and an instance of the control is created, this shouldn't affect your ability to evaluate the control. The registered version of the control is the same as the unregistered version except for a licence file that disables the nag screens. The licence file must be present at design time in order to disable nag screens.

In addition to registered copies of the control, the C++ source code to the control may also be ordered. The source code is supplied as is and is not supported in any way. Borland C++ 3.1 or Borland C++ 4.0 is required (the control is currently built with 4.0 but 3.1 may also be used with some bug patches), Microsoft source is not available (the code uses templates, which is a non-trivial problem for Microsoft users). If you are going to order the source code then you should probably also order the registered version of the control although it is not strictly necessary. The control determines if it is registered by looking in the directory from which it was loaded for the presence of a valid license file. The license is only distributed with the registered version of the control, without the registered control you will not be able to use the source without modifying it yourself to disable the nag screens. In order to build the control from the source you must also have a copy of vbapi.lib (it comes with the Visual Basic Professional Edition, for instance).

### PRICES...

Registered copies of the Early Morning Editor Control.....\$30.

Borland C++ Source Code.....\$30.

Shipping and Handling in US and Canada is \$4, elsewhere is \$6.

### CREDIT CARD ORDERS ONLY...

You can order with MC, Visa, Amex, or Discover from Public (software) Library by calling 800-2424-PsL or 713-524-6394 or by FAX to 713-524-6398 or by CIS Email to 71355,470. You can also mail credit card orders to PsL at P.O.Box 35705, Houston, TX 77235-5705.

To insure that you get the latest version, PsL will notify us the day of your order and we will ship the product directly to you.

### THE ABOVE NUMBERS ARE FOR ORDERS ONLY.

Any questions about the status of the shipment of the order, refunds, product details, technical support, etc, must be directed to 312-925-1628 or sent to Ted Stockwell, Early Morning Software, 3544 W 83rd Pl., Chicago IL 60652. Returns may be made to this address within 30 days of purchase.

**Clipboard**

A temporary storage location used to transfer text, graphics, and code.



**Insertion Point**

The location denoted by the CaretX and CaretY properties. If the control currently has the focus then this is the same as the caret position.

**Caret**

A flashing line, block, or bitmap that typically marks the location of the insertion point in a window's client

## Action Property

### Description

This property executes a desired action to be taken. Currently, this property is only used to interface with the Windows clipboard.

### Usage

**[form.]Editor.Action = value%**

### Remarks

This property can only be assigned. This property is not available at design time.

### Setting Description

- |    |                                                                                         |
|----|-----------------------------------------------------------------------------------------|
| 0  | No Action.                                                                              |
| 1  | Copy any selected text to the <u>Clipboard</u> .                                        |
| 2  | Paste a copy of any text on the Clipboard into the text at the <u>insertion point</u> . |
| 3  | Remove any selected text and put it on the Clipboard.                                   |
| 4  | Delete any selected text.                                                               |
| 5  | Wrap the selected text (see the <u>WrapX</u> and <u>WrapWholeWords</u> properties).     |
| 6  | Undo the last change.                                                                   |
| 7  | Redo the last change undone.                                                            |
| 8  | Empty the Undo buffer.                                                                  |
| 9  | Search for text (see the <u>SearchTarget Property</u> ).                                |
| 10 | Replace text (see the <u>SearchReplacement Property</u> ).                              |
| 11 | Repeat the last text search/replace operation.                                          |

### Data Type

Integer (Enumerated)

## CanRedo Property

### Description

If there are undone edit operations that can be redone then this property is True otherwise it is False. This property is not available at design time and read-only at run time.

### Usage

`[form.]Editor.CanRedo`

### Data Type

Integer (Boolean)

## CanUndo Property

### Description

If there are edit operations that can be undone then this property is True otherwise it is False. This property is not available at design time and read-only at run time.

### Usage

[form.]Editor.CanUndo

### Data Type

Integer (Boolean)

### See Also

[UndoLimit Property](#)

## Count Property

### Description

Specifies the number of lines of text in the control; not available at design time and read-only at run time.

### Usage

**[form.]Editor.Count**

### Data Type

Long

## CaretHeight Property

### Description

The height of the editor's caret in pixels. If CaretHeight is less than zero then the caret height will always be equal to the font height. The default value is -1. This property is typically used to change the shape of the caret to reflect the current insert mode.

### Usage

**[form.]Editor.CaretHeight[ = value ]**

### Data Type

Integer

### See Also

[CaretWidth Property](#)

## CaretWidth Property

### Description

The width of the editor's caret, in pixels. If CaretWidth is less than zero then the caret width will be equal to the maximum of 2 pixels or the width non-sizable window frames. The default value is -1. This property is typically used to change the shape of the caret to reflect the current insert mode.

### Usage

**[form.]Editor.CaretWidth[ = value ]**

### Data Type

Integer

### See Also

[CaretHeight Property](#)



## CaretX Property

### Description

The current X position of the insertion point. The X position of the insertion point may be changed by assigning to this property.

This property is not available at design time.

### Usage

**[form.]Editor.CaretX[ = value ]**

### Data Type

Long

### See Also

[CaretY Property](#)

## CaretY Property

### Description

The current Y position of the insertion point. The Y position of the insertion point may be changed by assigning to this property.

This property is not available at design time.

### Usage

**[form.]Editor.CaretY[ = value ]**

### Data Type

Long

### See Also

[CaretX Property](#)

## SelDefaultType Property

### Description

The Early Morning Editor Control supports line, column and stream blocks (stream blocks are the only block type in the standard Visual Basic text control). This property denotes the type of block created when a user interactively marks a block at run-time. Stream blocks are the default.

### Usage

[form.]Editor.SelDefaultType[ = value%]

Setting	Description
0	No blocks allowed.
1	Stream blocks are the default block type.
2	Line blocks are the default block type.
3	Column blocks are the default block type.

### Data Type

Integer (Enumerated)

## IsDirty Property

### Description

Denotes whether the text in the control has changed. The IsDirty property is set to True whenever the text in the control changes (whether the user changes the text or you change the text from code) and remains True until you set the property back to False. The IsDirty property is set to False whenever the FileOpen property is used to load a new file. This property is not available at design time.

### Usage

**[form.]Editor.IsDirty[ = value%]**

### Data Type

Integer (Boolean)

## FileOpen Property

### Description

Assigning to this property opens a text file and loads it into the control. After a file has been loaded this property contains the full path name of the loaded file. If there is already text in the control when you assign to this property the text will not be saved before the new text is loaded so if you want the text to be saved then you should be sure to save the text beforehand.

### Usage

**[form.]Editor.FileOpen[ = value%]**

### Data Type

String

### See Also

[FileSave Property](#)

## FileSave Property

### Description

Assigning to this property saves the text in the control to a file. This property is write-only at run-time.

### Usage

**[form.]Editor.FileSave = value%**

### Data Type

String

### See Also

[FileOpen Property](#)

## hFont Property

### Description

A handle to the font used by the editor control. This property is not available at design time and is read-only at run time.

### Usage

`[form.]Editor.hFont`

### Data Type

Integer

## InsertMode Property

### Description

When this property is True the control makes room for new text by moving existing text when False the control replaces existing text with incoming text. If you wish this property to be tied to the Insert key then you can trap Insert key presses with the KeyDown event and change the value of the InsertMode property whenever the the Insert key is pressed. The default is to insert text (True).

### Usage

[form.]Editor.IsDirty[ = value%]

### Data Type

Integer (Boolean)

### Example

This example uses the KeyDown event to trap Insert key presses and flip the InsertMode property...

```
Sub Editor1_KeyDown (KeyCode As Integer, Shift As Integer)
    ' If the Ins key is pressed and neither the Shift, Ctrl, nor Alt
    ' key is down then flip the insert mode property
    If KeyCode = KEY_INSERT And Shift = 0 Then
        Editor1.InsertMode = Not Editor1.InsertMode
    End If
End Sub
```



## LeftMargin, TopMargin, RightMargin and BottomMargin Properties

### Description

These properties are used to define margins (blanks areas) around the edge of the editor window. Margin units are in pixels. The most common use of these properties is to define a left margin so that text doesn't run all the way up to the left edge of the editor window.

### Usage

`[form.]Editor.LeftMargin[ = value%]`

### Data Type

Integer

## LeftHi Property

### Description

Used to find out what part of a line is marked (highlighted when a block is marked). This property equals the leftmost marked column (1 based) in the line to which the TextIndex refers. If no part of the line is marked then LeftHi equals zero. This property is read-only at run-time and is not available at design time.

### Usage

[form.]Editor.LeftHi

### Data Type

Long

### See Also

[RightHi Property](#), [TextIndex Property](#)

## LinesPerWindow and FullLinesPerWindow Properties

### Description

These properties are used to determine the number of lines of text that can be displayed in the editor's window. LinesPerWindow returns the number of lines that can be displayed including any partial lines at the bottom of the editor window. FullLinesPerWindow returns only the number of full lines that caan be displayed

These properties are not available at design time and are read-only at run time.

### Usage

**[form.]Editor.LinesPerWindow**

### Data Type

Integer

## ReadOnly Property

### Description

Makes the control read only, the user will not be able to alter the text in the control but will be able to scroll through the text, mark blocks, and copy text to the clipboard. The text in the control can still be altered from code, for instance, you can cut text using the Action property when ReadOnly is True. Setting this property to True makes the control read only, the default is False.

### Usage

**[form.]Editor.ReadOnly[ = value%]**

### Data Type

Integer(Boolean)

## Redraw Property

### Description

Used to reduce the amount of repainting and caret repositioning the control does during a long sequence of operations, thus making the control look like it's operating more smoothly. The default value for this property is True. Set this property to False to disable forced updating and caret repositioning during a long sequence of operations. Be sure to set this property back to True when finished. Not available at design time.

### Usage

**[form.]Editor.Redraw[ = value%]**

### Data Type

Integer(Boolean)

## RightHi Property

### Description

Used to find out what part of a line is marked (highlighted when a block is marked). This property equals the rightmost marked column (1 based) in the line to which the TextIndex refers. If no part of the line is marked then RightHi equals zero. This property is read-only at run-time and is not available at design time.

### Usage

[form.]Editor.RightHi

### Data Type

Long

### See Also

[LeftHi Property](#), [TextIndex Property](#)

## ScrollBars Property

### Description

Specifies whether an object has horizontal or vertical scroll bars; read-only at run time.

### Usage

**[form.]Editor.ScrollBars**

Setting	Description
0	(Default) None
1	Horizontal
2	Vertical
3	Both

### Data Type

Integer (Enumerated)

## SearchCaseSensitive Property

### Description

If set to True then text searches are case sensitive. The default is False.

### Usage

**[form.]Editor.SearchCaseSensitive[ = value ]**

### Data Type

Integer(Boolean)

### See Also

[SearchTarget Property](#), [SearchOrigin Property](#), [SearchTo Property](#), [SearchResult Property](#)



## SearchOrigin Property

### Description

Specifies where text searches begin. The default setting is zero.

### Setting Description

0 Searches start from current cursor position.  
1 Searches start from either beggin or end of file (depending on the setting of the SearchTo Property).

### Usage

[form.]Editor.SearchOrigin[ = value ]

### Data Type

Integer (Enumerated)

### See Also

[SearchCaseSensitive Property](#), [SearchResult Property](#), [SearchTarget Property](#), [SearchTo Property](#)

## SearchReplacement Property

### Description

This property specifies replacement text when the Action Property is used to replace text. This property should be set before using the Action Property to replace text.

### Usage

[form.]Editor.SearchReplacement[ = value ]

### Data Type

String

### Example

This example shows how to replace text...

```
Editor1.SearchTarget = "<PUT NEW TEXT HERE>"  
Editor1.SearchReplacement = "THIS IS THE NEW TEXT"  
Editor1.Action 10 'replace text
```

### See Also

[SearchTarget Property](#), [SearchOrigin Property](#), [SearchTo Property](#), [SearchResult Property](#)

## SearchResult Property

### Description

Check the value of this property after using the Action to search for text, it denotes whether text was found or not. This property is not available at design-time and is read only at run time.

### Setting Description

- |   |                                                                |
|---|----------------------------------------------------------------|
| 0 | Failure, no text was found.                                    |
| 1 | Success, at least one instance of the search target was found. |

### Usage

[form.]Editor.SearchResult

### Data Type

Integer (Enumerated)

### See Also

[Action Property](#), [SearchCaseSensitive Property](#), [SearchTarget Property](#), [SearchTo Property](#)

## SearchTarget Property

### Description

This property is used to specify the search target when the Action Property is used to search for text. This property should be set before using the Action Property to find text.

### Usage

**[form.]Editor.SearchTarget[ = value ]**

### Data Type

String

### Example

This example shows how to find the last occurrence of "Find Me"...

```
Editor1.SearchTarget = "Find Me"  
Editor1.SearchOrigin = 1 'entire scope  
Editor1.SearchTo = 0 'top of text  
Editor1.SearchCaseSensitive = False  
Editor1.SearchWholeWordsOnly = True  
Editor1.Action 9 'Search
```

```
If Editor1.SearchResult = 0 Then  
    MsgBox "Text Not Found"  
Endif
```

### See Also

[SearchCaseSensitive Property](#), [SearchOrigin Property](#), [SearchWholeWordsOnly Property](#), [SearchResult Property](#)

## SearchTo Property

### Description

Specifies where text searches end. The default setting is one.

### Setting Description

0	Top of text.
1	Bottom of text.

### Usage

[form.]Editor.SearchTo[ = value ]

### Data Type

Integer (Enumerated)

### See Also

[SearchCaseSensitive Property](#), [SearchOrigin Property](#), [SearchResult Property](#), [SearchTarget Property](#)

## SearchWholeWordsOnly Property

### Description

If set to True then text searches find occurrences that are words by themselves, and not part of a larger word. If set to False then text searches find all occurrences of the text. The default is False.

### Usage

[form.]Editor.SearchWholeWordsOnly[ = value ]

### Data Type

Integer(Boolean)

### See Also

[SearchCaseSensitive Property](#), [SearchOrigin Property](#), [SearchResult Property](#), [SearchTarget Property](#)

## SelBackColor Property

### Description

This property returns or changes the text color used to highlight selected text.

### Usage

**[form.]Editor.SelBackColor[ = value ]**

### Data Type

Color

### See Also

[SelForeColor](#)

## SelEndX, SelEndY, SelStartX, SelStartY Properties

### Description

These four properties denote the starting and ending points of the current block mark, if any, and may be used to size a block mark. These properties are not available at design time.

### Usage

[form.]Editor.SelEndX[ = value ]

### Data Type

Long

### See Also

[SelMark Property](#), [SelDefaultType Property](#)

### Example

The following example marks all the text without moving the caret...

```
Editor1.SelMark = 0 'unmark any existing block mark, if any
Editor.SelMark = 2 ' start a line block
Editor1.SelStartX = 1
Editor1.SelStartY = 1
Editor1.SelEndX = 1
Editor1.SelEndY = Editor.Count
```



## **SelfForeColor Property**

### **Description**

This property returns or changes the background color used to highlight selected text.

### **Usage**

**[form.]Editor.SelForeColor[ = value ]**

### **Data Type**

Color

### **See Also**

[SelBackColor](#)

## SelMark Property

### Description

Denotes the type of any current block mark. Can also be used to start a block mark or change the type of block. Not available at design time. The default block type is the block type specified by the SelDefaultType property.

### Usage

[form.]Editor.SelMark[ = value ]

Setting	Description
0	No block.
1	Stream block.
2	Line blocks.
3	Column block.

### Data Type

Integer (Enumerated)

### See Also

[SelEndX Property](#), [SelDefaultType Property](#)

## SelText Property

### Description

When read this property returns the highlighted text in the line to which the TextIndex property refers. When assigned this property replaces any current block mark with the given text, if there is no block mark then the given text is inserted at the insertion point. This property is not available at design time.

### Usage

[form.]Editor.SelText[ = value ]

### Data Type

String

### See Also

SelMark Property, SelDefaultType Property  
, TextIndex Property

## TabCount Property

### Description

Use this property to set the number of defined tab stops. After defining the number of tab stops with TabCount use TabMark to set each stop. TabDefaultWidth is used for tab stops that are not specifically defined. This property is not available at design time.

### Usage

**[form.]Editor.TabCount[ = value ]**

### Data Type

Integer

### See Also

[TabMark](#), [TabDefaultWidth](#)

## TabDefaultWidth Property

### Description

The default tab stop width, in pixels. By default, tab stops are set every TabDefaultWidth pixels apart. If TabCount and TabMark are used to define specific tab stops then the default tab stops are used only after the last tab mark specified by TabMark. Setting TabDefaultWidth to less than zero causes the default tab stop width to be 8 \* the average character width. The default value for TabDefaultWidth is -1.

### Usage

**[form.]Editor.TabDefaultWidth[ = value ]**

### Data Type

Integer

### See Also

[TabCount](#), [TabMark](#)

## TabMark Property

### Description

An array that defines tab stops, in pixels. The array ranges from zero to TabCount-1. This property is not available at design time.

The VB example below sets default tab marks at every 1 inch but also defines a tab stop at the half inch mark:

```
Editor1.TabDefaultWidth= 1440 / Screen.TwipsPerPixelX  
Editor1.TabCount= 1  
Editor.TabMark[0]= 720 / Screen.TwipsPerPixelX
```

### Usage

**[form.]Editor.TabDefaultWidth[ index ][ = value ]**

### Data Type

Long(array)

### See Also

[TabCount](#), [TabDefaultWidth](#)

## TextIndex Property

### Description

Used to refer to a line of text in the control. This property is used by other properties too determine what line to act on. The lines of text in a control are numbered beginning with one. This property is not available at design time.

### Usage

**[form.]Editor.TextIndex[ = value ]**

### Data Type

Long

## Text Property

### Description

When read this property returns the text in the line to which the TextIndex property refers. When assigned this property replaces the text in the line to which the TextIndex property refers with the given text. This property is not available at design time.

### Usage

**[form.]Editor.Text[ = value ]**

### Data Type

String

### See Also

[TextIndex Property](#)



## TopY Property

### Description

Denotes the line displayed at the top of the control's window. The line displayed at the top may be changed by assigning to this property. This property is not available at design time.

### Usage

**[form.]Editor.TopY[ = value ]**

### Data Type

Long

## UndoLimit Property

### Description

The maximum number of edit operations that can be undone. The default is 255. The higher this Property is set the more memory will be required to undo changes.

### Usage

**[form.]Editor.UndoLimit[ = value ]**

### Data Type

Integer

## WrapWholeWords Property

### Description

Used in conjunction with the Action Property to wrap the current text selection. This property denotes whether whole words should be kept whole. True means that words should be preserved. The default is False. This property is not available at design time. See the [WrapX Property](#) for an example.

### Usage

[form.]Editor.WrapWholeWords[ = value ]

### Data Type

Integer

### See Also

[Action Property](#), [WrapX Property](#)

## WrapX Property

### Description

Used in conjunction with the Action Property to wrap the current text selection. This property denotes the pixel position, starting from the left side of the text, where text is wrapped.

### Usage

[form.]Editor.WrapX[ = value ]

### Data Type

Integer

### See Also

[Action Property](#), [WrapWholeWords Property](#)

### Example

The following example wraps any current text selection, if any, to fit the editor window...

```
If Editor1.SelMark <> 0 Then
    Editor1.WrapX = Editor1.Width
    Editor1.WrapWholeWords = True
    Editor1.Action = 5 '...wrap the text
EndIf
```

## Change Event

### Description

Fired when the value of the IsDirty property changes.

### Syntax

Sub Editor1\_Change ()

### See Also

[IsDirty Property](#)

## CaretChange Event

### Description

Fired when the caret position changes. The OldCaretX and OldCaretY parameters specify the old caret position.

### Syntax

Sub Editor1\_CaretChange (OldCaretX As Long, OldCaretY as Long)

### See Also

[CaretX Property](#), [CaretY Property](#)

## BeginMessage Event, EndMessage Event, ProcessMessage Event

### Description

These are general purpose events that can be used to trap and process any message coming to the control. They are also good for detecting when some unique kind of event happens. BeginMessage is sent when a message is received and EndMessage is sent after the control has finished whatever processing it does, if any, in response to the message. The fProcessMessage parameter to the BeginMessage event can be used to disable the control's response to a message. If the fProcessMessage parameter is set to False then the control will not process the message, instead the ProcessMessage event is fired where you may write your own code to process the message. The control does not fire additional BeginMessage and EndMessage events in response to messages received during the processing of a BeginMessage, EndMessage, or ProcessMessage event.

### Syntax

Sub Editor1\_BeginMessage (HControl As Long, HWindow As Integer, Message As Integer, WParam As Integer, LParam As Long, fProcessMessage as Integer)

Sub Editor1\_EndMessage (HControl As Long, HWindow As Integer, Message As Integer, WParam As Integer, LParam As Long)

Sub Editor1\_ProcessMessage (HControl As Long, HWindow As Integer, Message As Integer, WParam As Integer, LParam As Long, MessageResult As Long)

### Remarks

If fProcessMessage is set to False during the processing of a BeginMessage event then the control will not perform the processing that it would normally perform in response to the message.

## FindWordBreak Event

### Description

Fired whenever the editor control must find the beginning of the next word in the control. This event will be fired during operations that require the text to be broken up into words, like when the left or right arrow keys are used in combination with the CTRL key or when the Action Property is used to wrap text while preserving words.

The default FindWordBreak function defines spaces, tabs, and the end points of lines as the breaks between words.

To define custom word breaks respond to this event by setting NextX and NextY to the X and Y position of the beginning of the first word after, but not at, position CurrentX and CurrentY.

### Syntax

Sub Editor1\_CaretChange (CurrentY As Long, CurrentX as Long, NextY As Long, NextX as Long)



## SearchReplaceConfirm Event

### Description

Fired whenever the editor control is about to replace text. Respond to this event by setting the Confirmation argument as follows:

#### Setting Description

- 0 CANCEL the replace operation.
- 1 YES, replace the text.
- 2 NO, don't replace the text.
- 3 Replace ONE instance, then stop.
- 4 Replace ALL instances.

The default response is 4, replace all instances.

Typically this event is used to display a dialog to the user.

### Syntax

Sub Editor1\_SearchReplaceConfirm (Confirmation as Integer)

## SelChange Event

### Description

Fired whenever the text selection changes. The OldSelStartX, OldSelStartY, OldSelEndX, OldSelEndY, and OldSelMark parameters specify the old text selection.

### Syntax

Sub Editor1\_SelChange (OldSelStartX As Long, OldSelStartY as Long, OldSelEndX As Long, OldSelEndY as Long, OldSelMark as Integer)

### See Also

[SelMark](#), [SelEndX](#), [SelEndY](#), [SelStartX](#), [SelStartY](#)

## TopYChange Event

### Description

Fired whenever the value of the TopY Property changes. The OldTopY specifies the old value of the TopY Property.

### Syntax

Sub Editor1\_TopYChange (OldTopY As Long)

### See Also

TopY



