Contents

Tips...

Whats New

V 1.6...

* WrapAutomatically Propery : When this propery is True text is automatically wrapped to fit within the boundary defined by the WrapX property. When WrapAutomatically is True text is automatically wrapped as it is entered or as the size of the editors window changes. When the value of WrapAutomatically is changed from True to False then any text in the control is changed to its unwrapped state.
* IsEndOfParagraph Propery : This property is for use when WrapAutomatically is True. This property denotes what lines are terminated by hard line breaks. This property can be used to changed what lines are terminated by hard line breaks.
* SelDragDropEnable Property : V1.6 of the Early Morning Editor has Drag and Drop text selections. This means that text can be moved by selecting some text, moving the mouse over the selected text and pressing the right mouse button and then dragging the text to a new location. Text can be copied in the same way by also holding the Ctrl key down. Drag and Drop text selections may be disabled by setting this property to False.
* Redraw Property : The Redraw Property has been enhanced by incorporating Undo/Redo control into it. When Redraw is set to False then all operations that are performed before Redraw is set back to True are grouped into one undo operation. That is, if immediately after setting Redraw to True the Action property is used to undo the last operation then all operations are undone up to the time immediately prior to setting the Redraw property to False.
* TextAll and SelTextAll Properties : These properties allow programmers to get text from and put text to an editor control in the same way as the native VB text control. However, these properties can never return more than 64K worth of text.
* FileOpen Property : This property has been enhanced so that it can also be used to load Unix text files (text files with lines terminated by only new line characters, not carriage return-new line sequences).
* FileInsert Property : This property is used to insert text from a disk file into an editor control, like pasting text into the control from the clipboard only its used to paste text from a text file.
* CountChange Event : This event is fired whenever the total number of lines in the control changes.
* Text Property : This property has be enhanced by making it respect the InsertMode property. Previously this property could only be used to overwrite the line of text in the control denoted by the TextIndex property. Now, when the InsertMode property is True, lines are inserted into the control at the position denoted by TextIndex. When the InsertMode property is False lines are overwritten just like before. Regardless of the value of the InsertMode property, when the TextIndex property is set to a position beyond the end of the current text lines are inserted at that point.

Handling Keyboard Input in C++ Programs…

Normally, Windows handles all arrow-key and TAB-key input to a control. By responding to the WM_GETDLGCODE message, an application can take control of a particular type of input and process the input itself. For instance, if you want the editor control to process TAB keys by putting a TAB into the control's text then you will have to respond to the WM_GETDLGCODE message and ask for the TAB keys.
This step is not neccessary when using Visual Basic, the C++ implementations of VBX support do not behave exaclty like Visual Basic in this regard.

Disabling Nag Screens in Applications Built with the 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. Displaying a nag screen 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 VB 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 when viewing the forms.

If you are working in C++ then you must load your project and REPLACE any existing editor controls with new controls.

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
FileInsert
FileOpen
FileSave
FontBold
FontItalic
FontName
FontSize
FontStrike
ForeColor
FullLinesPerWindow
Height
HelpContextID
hFont
hWnd
Index
InsertMode
IsDirty
IsEndOfParagraph
Left
LeftHi
LeftMargin
LinesPerWindow
LinkItem
LinkMode
LinkTimeOut
LinkTopic
MousePointer
Name
Parent
ReadOnly
Redraw
RightHi
RightMargin
ScrollBars
SearchCaseSensitive

SearchOrigin
SearchReplacement
SearchResult
SearchTarget
SearchTo
SearchWholeWordsOnly
SelDefaultType
SelDragDropEnable
SelEndX
SelEndY
SelForeColor
SelMark
SelStartX
SelStartY
SelText
SelTextAll
TabCount
TabDefaultWidth
TabMark
TabIndex
TabStop
Tag
Text
TextAll
TextIndex
Top
TopMargin
TopY
UndoLimit
Visible
Width
WrapAutomatically
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
CountChange
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++ 4.0 is required, 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. On CompuServe, questions and comments may be directed to Ted Stockwell at 74454,1002.

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 Clipboard.
2	Paste a copy of any text on the Clipboard into the text at the insertion point.
3	Remove any selected text and put it on the Clipboard.
4	Delete any selected text.
5	Wrap the selected text (see the WrapX and WrapWholeWords properties).
6	Undo the last change.
7	Redo the last change undone.
8	Empty the Undo buffer.
9	Search for text (see the SearchTarget Property).
10	Replace text (see the SearchReplacement Property).
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.
The system's window-border width or height can be retrieved by using the GetSystemMetrics function, specifying the SM_CXBORDER and SM_CYBORDER indices. Using the window-border width or height guarantees
that the caret will be visible on a high-resolution screen.

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.
The system's window-border width or height can be retrieved by using the GetSystemMetrics function, specifying the SM_CXBORDER and SM_CYBORDER indices. Using the window-border width or height guarantees
that the caret will be visible on a high-resolution screen.

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)

FileInsert Property

Description

Assigning to this property opens a text file and pastes it into the control, like pasting text from the clipboard. Like the FileOpen property, this property also handles Unix text files (text files with lines terminated by only new line characters, not carriage return-new line sequences). This property is write-only at run-time and is not available at design-time.

Usage

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

Data Type

String

See Also

FileSave Property

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.

This property can also be used to load Unix text files (text files with lines terminated by only new line characters, not carriage return-new line sequences).

Usage

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

Data Type

String

See Also

FileInsert Property, 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
```

IsEndOfParagraph Property

Description

This property is for use when WrapAutomatically is True. This property identifies lines that end with a hard line break. This property is False when the line referred to by the TextIndex property has been wrapped, otherwise it is True. The IsEndOfParagraph property may also be used to add or remove hard line breaks from line ends. This property is not available at design time.

For example, suppose you have two lines in a control before setting WrapAutomatically to True and that after setting WrapAutomatically to True there are four lines, the first line being split into three lines and the last line not being changed at all. When TextIndex equals 1 or 2 IsEndOfParagraph will return False. When TextIndex equals 3 or 4 IsEndOfParagraph will return True. If TextIndex is then set to 2, IsEndOfParagraph is set to True, and WrapAutomatically is set back to False then there will be three lines in the control, the first line being the concatenation of what were the 1st and 2nd lines and the last two lines remaining the same. Got that?

Naturally, when WrapAutomatically is False IsEndOfParagraph will always return True.

Usage

[form.]Editor. IsEndOfParagraph

Data Type

Integer (Boolean)

See Also

WrapAutomatically, WrapX

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. 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!!!!.
The Redraw Property also has Undo/Redo control built into it. When Redraw is set to False then all operations that are performed before Redraw is set back to True are grouped into one undo operation. That is, if immediately after setting redraw to True the Action property is used to undo the last operation then all operations are undone up to the time immediately prior to setting the Redraw property to False.
The default value for this property is True. 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 vaule 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 occurence 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

SelDragDropEnable Property

Description

The Early Morning Editor supports Drag and Drop text selections. This means that text can be moved by selecting some text, moving the mouse over the selected text and pressing the right mouse button and then dragging the text to a new location. Text can be copied in the same way by also holding the Ctrl key down. Drag and Drop text selections may be disabled by setting this property to False. Drag and Drop text selections only work within the current editor window, text selections may not be dragged to other editor windows. The default value of this property is True.

Usage

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

Data Type

Integer(Boolean)

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 = 1 ' start a stream block
Editor1.SelStartX = 1
Editor1.SelStartY = 1
Editor1.SelEndY = Editor.Count
Editor1.TextIndex = Editor.Count
Editor1.SelEndX =   Len( Editor1.Text) + 1
```

SelForeColor 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

SelTextAll Property

Description

When read this property returns all the highlighted text in an editor control as one string. 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. SelTextAll**[= value]**

Data Type

String

See Also

SelMark Property, SelDefaultType 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

This property is used to refer to a line of text in the control. This property is used by other properties to 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

See Also

Text Property

Text Property

Description

This property is used to either read, insert, or write over lines of text in the control.
When used to read lines of text this property returns the text in the line to which the TextIndex property refers.
When strings are assigned to this property the strings are either inserted into the control or used to overwrite a line of text depending on the value of the InsertMode property. When the InsertMode property is True, lines are inserted into the control at the position denoted by TextIndex. When the InsertMode property is False the line of text denoted by the TextIndex property is overwritten with the new text. Regardless of the value of the InsertMode property, when the TextIndex property is set to a position beyond the end of the current text lines are inserted at that point.
This property is not available at design time.

Usage

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

Data Type

String

See Also

TextIndex Property, TextAll Property

TextAll Property

Description

When read this property returns all the text in an editor control as one string. When assigned this property replaces all the text in an editor control with the given text. This property is not available at design time.

Usage

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

Data Type

String

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

WrapAutomatically Property

Description

When this propery is True text is automatically wrapped to fit within the boundary defined by the WrapX property. When WrapAutomatically is True text is automatically wrapped as it is entered, as the size of the editors window changes, or as the size of the font changes. When the value of WrapAutomatically is changed from True to False then any text in the control is changed to its unwrapped state. When the FileSave property is used to save text in a control that has WrapAutomatically set to True then the text is saved in its unwrapped state. The default value for WrapAutomatically is False.

Note:Since the editor will not undo changes to its window size or changes in its font size it will not undo any changes to its text before these events when WrapAutomatically is True.

Usage

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

Data Type

Integer(Boolean)

See Also

WrapX Property, WrapWholeWords, IsEndOfParagraph

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 True. 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. If WrapX is greater than or equal to zero then WrapX denotes the exact pixel position where text is wrapped. If WrapX is equal to -1 then text is wrapped to fit the current window. When text is wrapped to fit the window the current margin settings are taken into account as well as the current width of the caret (enough room is left at the end of a line for the caret). The default is -1.

Usage

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

Data Type

Integer

See Also

Action Property, WrapWholeWords Property

Example

The following example wraps any current text selection so that no line is longer than 6 logical inches...

```
Const TWIPS_PER_LOGICAL_INCH% = 1440
If Editor1.SelMark <> 0 Then   if there is text selected...
        Editor1.WrapX = (6 * TWIPS_PER_LOGICAL_INCH) / Screen.TwipsPerPixelX
        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.

CountChange Event

Description

Fired whenever the total nuber of lines in the editor control changes.

Syntax

Sub Editor1_CountChange (OldCount as Long)

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 paragraphs 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 (CurrentX as Long, CurrentY As Long, NextX as Long, NextY 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