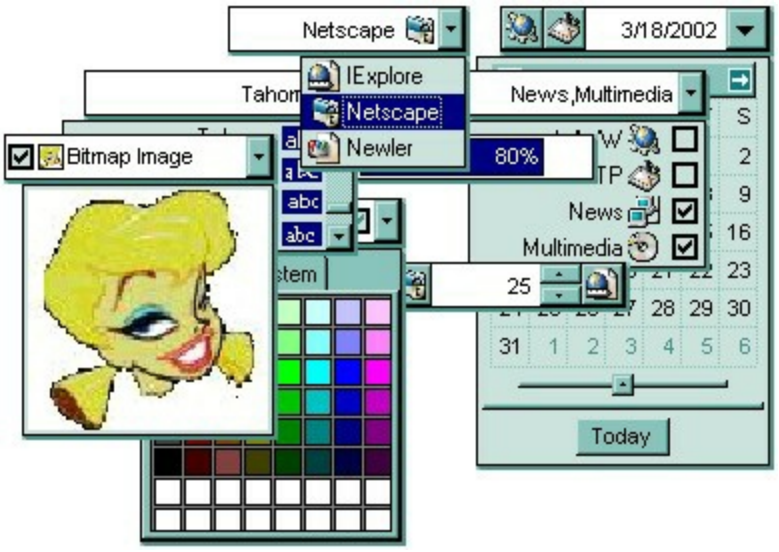
ExEditors

The Exontrol's ExEditors Library contains 16 data edit controls. Each editor can have a three-state check box associated and an unlimited number of buttons on the left or right side. Each button can display an icon or a picture and can have its own tool tip. Most of the editors are mask based. COM/ActiveX or .NET Assembly available as separate products.

The suite of editors includes:

- None: Provides a standard label control
- **Edit**: Provides a standard text edit control
- **DropDown**: Provides an intuitive way for users to select values from a list presented in a drop down window. Each item can display an icon and a caption. The DropDown allows custom strings as well.
- **DropDownList**: Provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. Each item can have an icon associated. The DropDownList editor has no edit control associated.
- **Spin**: Allows your users to view and change numeric values using a up/down button.
- **Memo**: Provides a multiline edit control.
- **CheckList**: Provides an intuitive way to view and combine one or more values from a pre-defined list presented in a drop down window. Each item has a check box attached, and can display an icon and a caption.
- **Date**: Provides an efficient way for selecting dates for a drop down calendar.
- **Mask**: You can use the Mask editor to enter any data that includes literals and requires a mask to filter characters during data input.
- **Color,ColorList**: Provides an elegant way for selecting colors from a dropdown window. The Color drop down window contains two tabs the pallette tab shows a grid of colors, while the system tab shows the current windows color constants.
- **Font**: Presents the list of fonts into a drop down window. Each item into the drop down window has a preview image for the font selected.
- **Picture**: Provides an elegant way for displaying fields of Picture type. The Picture data editor can display embedded images stored in OLE Object fields.
- **Button**: Associates single button to a mask edit control.
- **ProgressBar**: Displays your value using a progress bar style.
- **PickEdit**: Provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. Each item can have an icon associated.
- **LinkEdit**: Allows your application to edit and display hyperlink addresses.



How to get support?

To keep your business applications running, you need support you can count on.

Here are few hints what to do when you're stuck on your programming:

- Check out the samples - they are here to provide some quick info on how things should be done
- Check out the how-to questions using the eXHelper tool
- Check out the help - includes documentation for each method, property or event
- Check out if you have the latest version, and if you don't have it send an update request here.
- Submit your problem(question) here.

Don't forget that you can contact our development team if you have ideas or requests for new components, by sending us an e-mail at support@exontrol.com (please include the name of the product in the subject, ex: exgrid) . We're sure our team of developers will try to find a way to make you happy - and us too, since we helped.

Regards,
Exontrol Development Team

https://www.exontrol.com

constants AlignmentEnum

Specifies the alignment for the icon and the caption of the editor.

Name	Value	Description
LeftAlignment	0	The caption and the icon are left aligned
CenterAlignment	1	The caption is center aligned, the icon is left aligned
RightAlignment	2	The caption and the icon are right aligned

constants AppearanceEnum

The AppearanceEnum enumeration is used to specify the appearance of the control's borders. Use the Appearance property to specify the control's border.

Name	Value	Description
None2	0	No border
Flat	1	Flat border
Sunken	2	Sunken border
Raised	3	Raised border
Etched	4	Etched border
Bump	5	Bump border

constants BackgroundPartEnum

The BackgroundPartEnum type indicates parts in the control. Use the Background property to specify a background color or a visual appearance for specific parts in the control. A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Name	Value	Description
exDropDownButtonUp	0	Specifies the visual appearance for the drop down button, when it is up.
exDropDownButtonDown	1	Specifies the visual appearance for the drop down button, when it is down.
exButtonUp	2	Specifies the visual appearance for the button inside the editor, when it is up.
exButtonDown	3	Specifies the visual appearance for the button inside the editor, when it is down.
exDateHeader	8	Specifies the visual appearance for the header in a calendar control.
exDateTodayUp	9	Specifies the visual appearance for the today button in a calendar control, when it is up.
exDateTodayDown	10	Specifies the visual appearance for the today button in a calendar control, when it is down.
exDateScrollThumb	11	Specifies the visual appearance for the scrolling thumb in a calendar control.
exDateScrollRange	12	Specifies the visual appearance for the scrolling range in a calendar control.
exDateSeparatorBar	13	Specifies the visual appearance for the separator bar in a calendar control.
exDateSelect	14	Specifies the visual appearance for the selected date in a calendar control.
exSpinUpButtonUp	22	Specifies the visual appearance for the up spin button when it is not pressed.
exSpinUpButtonDown	23	Specifies the visual appearance for the up spin button when it is pressed.
exSpinDownButtonUp	24	Specifies the visual appearance for the down spin

		button when it is not pressed.
exSpinDownButtonDown	25	Specifies the visual appearance for the down spin button when it is pressed.
exToolTipAppearance	64	Specifies the visual appearance of the borders of the tooltips.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.

constants CheckStateEnum

Specifies whether the editor's checkbox displays two or three states. The PartialCheck property specifies whether the checkbox of the editor displays two states or three states. Use the HasCheckBox property to assign a checkbox to an editor.

Name	Value	Description
Unchecked	0	The associated check box is unchecked.
Checked	1	The associated check box is checked.
PartialChecked	2	The associated check box is partially checked.

constants EditorOptionEnum

Specifies different options for a built-in editor. The Option property specifies the editor's options. The following options are supported:

Name	Value	Description
exMemoHScrollBar	1	Adds the horizontal scroll bar to a MemoType or MemoDropDownType editor. By default, the Editor.Option(exMemoHScrollBar) is False. (boolean expression
exMemoVScrollBar	2	Adds the vertical scroll bar to a MemoType or MemoDropDownType editor. By default, the Editor.Option(exMemoVScrollBar) is False. (boolean expression)
exEditRight	3	Right-aligns text in a single-line or multiline edit control.
exEditDecimalSymbol	4	Specifies the symbol that indicates the decimal values while editing a floating point number.
exEditPassword	5	Displays all characters as an asterisk (*) as they are typed into the edit control.
exEditPasswordChar	6	Specifies the password char for an edit control.
exEditLimitText	7	Limits the length of the text that the user may enter into an edit control.
exProgressBarBackColor	8	Specifies the background color for a progress bar editor.
exProgressBarAlignment	9	Specifies the alignment of the progress bar caption.
exProgressBarMarkTicker	10	Marks the outside rectangle of a progress bar.
exDateAllowNullDate	11	Allows empty date to an DateType editor.
exEditSelStart	48	Sets the starting point of text selected, when an EditType editor is opened. By default, the exEditSelStart property is 0. If the exEditSelStart property is 0, the text gets selected from the first character. If the exEditSelStart property is -1, the cursor is placed at the end of the text. (long expression)
		Sets the number of characters selected, when an EditType editor is opened. By default, the

exEditSelLength	49	exEditSelLength property is -1. If the exEditSelLength is 0, no text is selected, instead the exEditSelStart changes the position of the cursor. If the exEditSelLength property is -1, the text from the exEditSelStart position to the end gets selected. (long expression)
exEditAllowOverType	200	Specifies whether the editor supports overtype mode. The option is valid for EditType and MemoType editors. (boolean expression, by default it is False).
exEditOverType	201	Retrieves or sets a value that indicates whether the editor is in insert or overtype mode. The option is valid for EditType and MemoType editors. (boolean expression, by default it is False).
exDateTodayCaption	202	Specifies the caption for the 'Today' button in a DateType editor.
exDateMonths	203	Specifies the name for months to be displayed in a DateType editor.
exDateWeekDays	204	Specifies the shortcut for the weekdays to be displayed in a DateType editor.
exDateFirstWeekDay	205	Specifies the first day of the week in a DateType editor.
exDateShowTodayButton	206	Specifies whether the 'Today' button is visible or hidden in a DateType editor.
exDateMarkToday	207	Gets or sets a value that indicates whether the today date is marked in a DateType editor.
exDateShowScroll	208	Specifies whether the years scroll bar is visible or hidden in a DateType editor.
exDateWeeksHeader	209	Sets or gets a value that indicates whether the weeks header is visible or hidden in a drop down date editor.
exSpinStep	210	Specifies the proposed change when user clicks a spin control.
exDropDownBackColor	55	exDropDownBackColor. Specifies the drop down's background color.
exDropDownForeColor	56	exDropDownForeColor. Specifies the drop down's foreground color.

constants EditTypeEnum

The control's EditType property defines the control's type. The Editor object supports the following types:

Name	Value	Description
NoneType	0	Provides a standard label control. The Value property defines the caption in the label.
EditType	1	Provides a standard text edit control. The Value property specifies the caption for the edit control.
DropDownType	2	Provides an intuitive way for users to select values from a list presented in a drop down window. Each item can display an icon and a caption. The DropDown allows custom strings as well. The DropDownType editor associates a standard text edit field too. Use the AddItem method to add new items in the control's predefined list. The editor displays the Value property.
DropDownListType	3	Provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. The DropDownList editor doesn't associate a standard edit control. Use the AddItem method to add new items in the control's predefined list. The control displays the caption of the item that has the value equal with the Value property. If the control cannot find an item with specified value, nothing is displayed.
SpinType	4	Allows your users to view and change numeric values using a up/down button. The Value property determines the caption being displayed in the edit control.
MemoType	5	Provides a multiline edit control. The Value property provides the text inside a Memo editor.
CheckListType	6	Provides an intuitive way to view and combine one or more values from a pre-defined list presented in a drop down window. Each item has a check box attached, and can display an icon and a caption. The control displays the items that are a bit combination of Value property.
		Provides an efficient way for selecting dates for a

DateType	7	drop down calendar. The Value property specifies the date being selected.
MaskType	8	You can use the Mask editor to enter any data that includes literals and requires a mask to filter characters during data input. The Value property specifies the text of the Mask control, and the Mask property determines the control's mask.
ColorType	9	Provides an elegant way for selecting colors from a dropdown window. The Color drop down window contains two tabs the pallette tab shows a grid of colors, while the system tab shows the current windows color constants. The Value property specifies the color being selected.
FontType	10	Presents the list of fonts into a drop down window. Each item into the drop down window has a preview image for the font selected. The Value property specifies the name of the font being selected.
PictureType	11	Provides an elegant way for displaying fields of Picture type. The Picture data editor can display embedded images stored in OLE Object fields. The Value property specifies the OLE object, as they are passed from a recordset, or a Picture object.
ButtonType	12	Associates single button to a mask edit control. The Value property determines the caption of the mask control.
ProgressBarType	13	Displays your value using a progress bar style. The Value property specifies the value being displayed as %.
PickEditType	14	Provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. Each item can have an icon associated. Use the AddItem method to add new items in the control's predefined list. The control displays the caption of the item that has the value equal with the Value property. If the control cannot find an item with specified value, nothing is displayed.
LinkEditType	15	Allows your application to edit and display hyperlink addresses. The Value property specifies the

		hyperlink address being displayed.
ColorListType	16	The editor hosts a predefined list of colors. By default. the following colors are added: Black, White, Dark Red, Dark Green, Dark Yellow, Dark Blue, Dark Magenta, Dark Cyan, Light Grey, Dark Grey, Red, Green, Yellow, Blue, Magenta, Cyan. The AddItem method adds new colors to the editor . The Value property specifies the color being selected.

constants InplaceAppearanceEnum

Specifies the control's appearance.

Name	Value	Description
NoApp	0	No border
FlatApp	1	Flat
SunkenApp	2	Sunken
RaisedApp	3	Raised
EtchedApp	4	Etched
BumpApp	5	Bump
ShadowApp	6	Shadow
InsetApp	7	Inset
SingleApp	8	Single

constants NumericEnum

Use the Numeric property to specify the format of numbers when editing a field.

Name	Value	Description
exInteger	-1	Allows editing numbers of integer type. The format of the integer number is: **[+/-]digit**, where **digit** is any combination of digit characters. This flag can be combined with exDisablePlus, exDisableMinus or exDisableSigns flags. For instance, the 0x3FF (hexa representation, 1023 decimal) value indicates an integer value with no +/- signs.
exAllChars	0	Allows all characters. No filtering.
exFloat	1	Allows editing floating point numbers. The format of the floating point number is: **[+/-]digit[.digit[[e/E/d/D][+/-]digit]]**, where digit is any combination of **digit** characters. Use the exEditDecimalSymbol option to assign a new symbol for '**.**' character (decimal values). This flag can be combined with exDisablePlus, exDisableMinus or exDisableSigns flags.
exFloatInteger	2	Allows editing floating point numbers without exponent characters such as e/E/d/D, so the accepted format is **[+/-]digit[.digit]**. Use the exEditDecimalSymbol option to assign a new symbol for '**.**' character (decimal values). This flag can be combined with exDisablePlus, exDisableMinus or exDisableSigns flags.
exDisablePlus	256	Prevents using the + sign when editing numbers. If this flag is included, the user can not add any + sign in front of the number.
exDisableMinus	512	Prevents using the - sign when editing numbers. If this flag is included, the user can not add any - sign in front of the number.
exDisableSigns	768	Prevents using the +/- signs when editing numbers. If this flag is included, the user can not add any +/- sign in front of the number. For instance exFloatInteger + exDisableSigns allows editing floating points numbers without using the exponent and plus/minus characters, so the allowed format is

digit[.digit]

constants UIVisualThemeEnum

The UIVisualThemeEnum expression specifies the UI parts that the control can shown using the current visual theme. The UseVisualTheme property specifies whether the UI parts of the control are displayed using the current visual theme.

Name	Value	Description
exNoVisualTheme	0	exNoVisualTheme
exDefaultVisualTheme	16777215	exDefaultVisualTheme
exButtonsVisualTheme	4	exButtonsVisualTheme
exCalendarVisualTheme	8	exCalendarVisualTheme
exSpinVisualTheme	32	exSpinVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exProgressVisualTheme	128	exProgressVisualTheme

Appearance object

The component lets the user changes its visual appearance using **skins**, each one providing an additional visual experience that enhances viewing pleasure. Skins are relatively easy to build and put on any part of the control. The Appearance object holds a collection of skins. The Appearance object supports the following properties and methods:

Name	Description
Add	Adds or replaces a skin object to the control.
Clear	Removes all skins in the control.
Remove	Removes a specific skin from the control.

method Appearance.Add (ID as Long, Skin as Variant)

Adds or replaces a skin object to the control.

Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
Skin as Variant	A string expression that indicates: 1. an Windows XP Theme part, it should start with "XP:". For instance the "**XP:Header 1 2**" indicates the part 1 of the Header class in the state 2, in the current Windows XP theme. In this case the format of the Skin parameter should be: "XP: Control/ClassName Part State" where the ClassName defines the window/control class name in the Windows XP Theme, the Part indicates a long expression that defines the part, and the State indicates the state like listed at the end of the document. This option is available only on Windows XP that supports Themes API. 2. a copy of another skin with different coordinates, if it begins with "CP:" . For instance, you may need to display a specified skin on a smaller rectangle. In this case, the string starts with "CP:", and contains the following "CP:n l t r b", where the n is the identifier being copied, the l, t, r, and b indicate the left, top, right and bottom coordinates being used to adjust the rectangle where the skin is displayed. For instance, the "**CP:1 4 0 -4 0**", indicates that the skin is displayed on a smaller rectangle like follows. Let's say that the control requests painting the {10, 10, 30, 20} area, a rectangle with the width of 20 pixels, and the height of 10 pixels, the skin will be displayed on the {14,10,26,20} as each coordinates in the "CP" syntax is added to the displayed rectangle, so the skin looks smaller. This way you can apply different effects to your objects in your control. The following screen shot shows the control's header when using a "CP:1 -6 -6 6 6", that displays the original skin on larger rectangles .

3. the path to the skin file (*.ebn). The Exontrol's exButton component installs a skin builder that should be used to create new skins
4. the BASE64 encoded string that holds a skin file (*.ebn). Use the Exontrol's exImages tool to build BASE 64 encoded strings on the skin file (*.ebn) you have created. Loading the skin from a file (eventually uncompressed file) is always faster then loading from a BASE64 encoded string

A byte[] or safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use this option when using the EBN file directly in the resources of the project. For instance, the VB6 provides the LoadResData to get the safe array o bytes for specified resource, while in VB/NET or C# the internal class Resources provides definitions for all files being inserted. (ResourceManager.GetObject("ebn", resourceCulture)).

Return	Description
Boolean	A Boolean expression that indicates whether the new skin was added or replaced.

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (*.ebn) assigned to a part of the control, when the "XP:" prefix is not specified in the Skin parameter (available for Windows XP systems). By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part. Use the Remove method to remove a specific skin from the control. Use the Clear method to remove all skins in the control. Use the Refresh method to refresh the control.

method Appearance.Clear ()

Removes all skins in the control.

Type	Description

method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

Type	Description
ID as Long	A Long expression that indicates the index of the skin being removed.

Use the Remove method to remove a specific skin. The identifier of the skin being removed should be the same as when the skin was added using the Add method. Use the Clear method to clear all skins from the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Editor object

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {ED8B26D2-1855-42D5-B622-FEE85395DFB3}. The object's program identifier is: "Exontrol.Editor". The /COM object module is: "ExEditors.dll"

The ExEditors Library, a 100% ATL based component suite, contains 16 data edit controls in a single ActiveX control. Each editor can have a three-state check box associated and an unlimited number of buttons on the left or right side. Each button can display an icon or a picture and can have its own tool tip. Most of the editors are mask based. The Editor object supports the following methods and properties:

Name	Description
AddButton	Adds a new button to the editor with the Key and aligned to the left or right side of the editor.
AddItem	Adds a new item to editor's predefined list.
Alignment	Retrieves or sets a value that indicates the caption's alignment.
Appearance	Retrieves or sets the control's appearance.
BackColor	Retrieves or sets the control's background color.
Background	Returns or sets a value that indicates the background color for parts in the control.
ButtonWidth	Retrieves or sets a value that indicates the button's width.
CheckState	Retrieves or sets a value that indicates the check box's state.
ClearButtons	Clears the buttons collection.
ClearItems	Clears the items collection.
DropDown	Shows up the control's drop down list.
DropDownAlignment	Retrieves or sets a value that indicates the item's alignment into control's drop down list.
DropDownAutoWidth	Retrieves or sets a value that indicates whether the control's drop down list width is automatically computed to fit the entire list.
DropDownMinWidth	Specifies the minimum drop down list width while the DropDownAutoWidth is False.
DropDownRows	Retrieves or sets a value that indicates the maximum number of rows into a drop down list.
	Retrieves or sets a value that indicates whether the

DropDownVisible	control's drop down window is shown or hidden.
EditText	Specifies the editor's caption.
EditType	Retrieves or sets a value that indicates the type of the contained editor.
Enabled	Retrieves or sets a value that indicates whether the control is enabled or disabled.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
FindItem	Finds an item given its value or caption.
Font	Retrieves or sets the control's font.
ForeColor	Retrieves or sets the control's foreground color.
HasCheckBox	Retrieves or sets a value that indicates whether the check box is visible or hidden.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
Images	Sets the control's handle image list.
ImageSize	Retrieves or sets the size of icons the control displays.
ItemCaption	Gets the item's caption giving its index.
ItemCount	Counts the items in the collection.
Mask	Retrieves or sets a value that indicates the mask used by the editor.
MaskChar	Retrieves or sets a value that indicates the character used for masking.
Numeric	Specifies whether the editor enables numeric values only.
Option	Specifies an option for the editor.
PartialCheck	Retrieves or sets a value that indicates whether the associated check box has two or three states.
PopupAppearance	Retrieves or sets a value that indicates the popup window's appearance.
ReadOnly	Retrieves or sets a value that indicates whether the control is read-only.
Refresh	Refreshes the control.
RemoveButton	Removes a button given its key.

RemoveItem	Removes the item from editor's list given its value.
ReplaceIcon	Adds a new icon, replaces an icon or clears the control's image list.
SelBackColor	Retrieves or sets a value that indicates the selection background color.
SelForeColor	Retrieves or sets a value that indicates the selection foreground color.
ShowImageList	Specifies whether the control's image list window is visible or hidden.
SortItems	Sorts the list of items in the editor.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipPopDelay	Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.
ToolTipWidth	Specifies a value that indicates the width of the tooltip window, in pixels.
UseVisualTheme	Specifies whether the control uses the current visual theme to display certain UI parts.
Value	Retrieves or sets the control's value
Version	Retrieves the control's version.
VisualAppearance	Retrieves the control's appearance.

method Editor.AddButton (Key as Variant, [Image as Variant], [Align as Variant], [ToolTip as Variant], [ToolTipTitle as Variant])

Adds a new button to the editor with the Key and aligned to the left or right side of the editor.

Type	Description
Key as Variant	A string expression that indicates the button's key. The Key parameter is passed to the ButtonClick event when user clicks the button.
Image as Variant	A long expression that indicates the index of icon being displayed into the button, or a Picture object that contains the buttons' image. 0 means no icon.
Align as Variant	An AlignmentEnum expression that indicates the button's alignment in the editor's client area.
ToolTip as Variant	A string expression that indicates the button's tool tip. The tooltip shows up when the user moves the cursor over the button.
ToolTipTitle as Variant	A string expression that indicates the button's tool tip title.

The AddButton method adds new buttons to the control. Use the ClearButtons to clear the buttons collection. Use the RemoveButton method to remove a button given its key. The ButtonClick event notifies your application that the user clicks a button in the editor. Use the ButtonWidth property to specify the width of the button , in pixels.

The following sample adds two left aligned buttons, and three right aligned buttons:

```
Editor1.AddButton "Add", 1, LeftAlignment, "Click here to add new items", "Add"
Editor1.AddButton "Load", 2, LeftAlignment, "Click here to load a file", "Load"

Editor1.AddButton "Save", 3, RightAlignment, "Click here to save the file.", "Save"
Editor1.AddButton "Remove", 4, RightAlignment, "Click here to remove the current item", "Remove"
Editor1.AddButton "Clear", 5, RightAlignment, "Click here to clear the items", "Clear"
```

The following sample adds a new button using a picture file:

```
Editor1.AddButton "Save", LoadPicture("save.gif"),  RightAlignment, "Click here to save the file.", "Save"
```

method Editor.AddItem (Value as Long, Caption as String, [Image as Variant])

Adds a new item to editor's predefined list.

Type	Description
Value as Long	A long expression that indicates the item's value.
Caption as String	A string expression that indicates the item's caption. The Caption property supports built-in HTML format.
Image as Variant	A long expression that indicates the index of the icon being displayed in the item.

The AddItem method adds new items for editors of the following type: DropDown, DropDownList, CheckList and PickEdit. Use the EditType property to change the editor's type. Adding an item with the same value removes the old one and add the new one. So, please make sure that you are using different values. If the editor's type is CheckList the Value property is a bit combination of checked items in the drop down list. If the editor's type is DropDowList or PickEdit the editor's value is the item's value. If the editor's type is DropDown, the editor's Value is the item's caption. Use the RemoveItem method to remove an item from the editor's predefined list. Use the ClearItems method to clear the entire list of predefined items. Use the Refresh method to refresh the control. Use the FindItem property to get the caption of the item's value. The ItemCount property counts the number of items in the control's predefined list. The ItemCaption property gives the caption of an item giving its index. Use the SortItems method to sort the list of items.

The Caption parameter supports built-in HTML format like follows:

-   **bold**
- <u> underline </u>
- <s> ~~strikeout~~ </s>
- *italic*
- <fgcolor = FF0000> fgcolor </fgcolor>
- <bgcolor = FF0000> bgcolor </bgcolor>
-
 breaks a line.
- <solidline> draws a solid line
- <dotline> draws a dotted line
- <upline> draws the line to the top of the text line
- <r> aligns the rest of the text line to the right side.
- ****text displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, <font

;12>bit displays the bit text using the current font, but with a different size.

- **&** glyph characters as **&amp** (&), **&lt** (<), **&gt** (>), **&qout** ("), **&#number**, For instance, the &#8364 displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display bold in HTML caption you can use bold

Newer HTML format supports subscript and superscript like follows:

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: *"Text with <**off** 6>subscript"* displays the text such as: Text with $_{\text{subscript}}$ The *"Text with <**off** -6>superscript"* displays the text such as: Text with $^{\text{subscript}}$

Also, newer HTML format supports decorative text like follows:

- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "*<**gra** FFFFFF;1;1>gradient-center</**gra**>*" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "*<**out** 000000> <fgcolor=FFFFFF>outlined</fgcolor></**out**>*" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray,

width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "*<**sha**>shadow</**sha**>*" generates the following picture:

or "*<**sha** 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></**sha**>*" gets:

The following sample shows how to combine the value for a CheckList type editor:

```
With Editor1
   .AddItem 1, "CanLink", 1
   .AddItem 2, "CanShare", 2
   .AddItem 4, "CanMove", 3
   .AddItem 8, "CanRestore", 3
   .EditType = CheckList
   .Value = 1 + 4 ' CanLink + CanMove
End With
```

property Editor.Alignment as AlignmentEnum

Retrieves or sets a value that indicates the caption's alignment.

Type	Description
AlignmentEnum	An AlignmentEnum value that indicates the caption's alignment.

If the Alignment is LeftAlignment the icon and the caption are left aligned. If the Alignment is RightAlignment the icon and the caption are right aligned. If the Alignment is CenterAlignment the caption is center aligned, and the icon is left aligned. Use the DropDownAlignment property to align the items into a drop down window

property Editor.Appearance as AppearanceEnum

Retrieves or sets the control's appearance

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the Appearance collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. ***The Client object in the skin, defines the client area of the control. The control's client area are is displayed in the skin's client area. The skin may contain transparent objects, and so you can define round corners. The normal.ebn file contains such of objects. Use the eXButton's Skin builder to view or change this file***

Use the Appearance property to hide the control's border. Use the PopupAppearance property to change the appearance for the drop down window. Use the BackColor property to specify the control's background color. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips

property Editor.BackColor as Color

Retrieves or sets the control's background color.

Type	Description
Color	A color expression that indicates the control's background color.

Use the BackColor property to change the control's background color. The BackColor property does not change the drop down window's background color. Use the ForeColor property to change the control's foreground color.

property Editor.Background(Part as BackgroundPartEnum) as Color

Returns or sets a value that indicates the background color for parts in the control.

Type	Description
Part as BackgroundPartEnum	A BackgroundPartEnum expression that indicates a part in the control.
Color	A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

The Background property specifies a background color or a visual appearance for specific parts in the control. If the Background property is 0, the control draws the part as default. Use the Add method to add new skins to the control. Use the Remove method to remove a specific skin from the control. Use the Clear method to remove all skins in the control. Use the Refresh method to refresh the control.

property Editor.ButtonWidth as Long

Retrieves or sets a value that indicates the button's width.

Type	Description
Long	A long expression that indicates the button's width in pixels.

By default, the ButtonWidth property is GetSystemMetrics(SM_CXHSCROLL). The GetSystemMetrics function retrieves various system metrics (widths and heights of display elements) and system configuration settings. Use the ButtonWidth property to specify the width for all buttons. Use the DropDownVisible to show or hide the drop down button. If the ButtonWidth property is 0, no buttons are displayed (including the drop down button).

property Editor.CheckState as CheckStateEnum

Retrieves or sets a value that indicates the check box's state.

Type	Description
CheckStateEnum	A CheckStateEnum expression that indicates the state of the control's check box.

Use the CheckState property to change the control's check box state. The CheckStateChanged event notifies your application that the CheckState property is changing. Use the HasCheckBox property to show or hide the control's check box. Use the PartialCheck property to allow two or three states for the check box.

method Editor.ClearButtons ()

Clears the buttons collection.

Type	Description

Use the RemoveButton method to remove a button given its key.

method Editor.ClearItems ()

Clears the items collection.

Type	Description

The ClearItems removes all predefined items of the control. Use the AddItem method to add a new value to the control's predefined list. Use the RemoveItem method to remove a specific item. Use the FindItem property to retrieve the caption of an item in the predefined list of items.

method Editor.DropDown ()

Shows up the control's drop down list.

Type	Description

Use the DropDown property to programmatically display the control's drop down window. Use the EditType property to change the control's type. If the DropDownVisible is False, no drop down window is shown. The DropDown property shows the drop down window only if the control's EditType property is one of the following: DropDown, DropDownList, CheckList, Date, Color, Font, Picture, ColorList, PickEdit.

property Editor.DropDownAlignment as AlignmentEnum

Retrieves or sets a value that indicates the item's alignment into control's drop down list.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the item's alignment.

The DropDownAlignment property specifies the alignment for the items in a drop down window. The DropDownAlignment property has effect only if the EditType property is one of the following: DropDown, DropDownList, CheckList, PickEdit and Font. Use the Alignment property to change the caption's alignment.

property Editor.DropDownAutoWidth as Boolean

Retrieves or sets a value that indicates whether the control's drop down list width is automatically computed to fit the entire list.

Type	Description
Boolean	A boolean expression that indicates whether the control's drop down window width is automatically computed to fit the entire list.

By default, the DropDownAutoWidth property is False. If the DropDownAutoWidth property is True, the DropDownMinWidth property has no effect. The DropDownAutoWidth property has effect if the control's EditType property is one of the followings: DropDown, DropDownList, CheckList, Font, PickEdit or ColorList.

property Editor.DropDownMinWidth as Long

Specifies the minimum drop down list width while the DropDownAutoWidth property is False.

Type	Description
Long	A long expression that indicates the minimum width of the drop down window, in pixels.

By default, the DropDownMinWidth property is 64. Use the DropDownMinWidth property to specify the minimum width for the drop down window. The DropDownMinWidth property has effect only if the DropDownAutoWidth property is False. The DropDownMinWidth property has effect only if the control's EditType property is one of the followings: DropDown, DropDownList, CheckList, PickEdit or ColorList.

property Editor.DropDownRows as Long

Retrieves or sets a value that indicates the maximum number of rows into a drop down list.

Type	Description
Long	A long expression that indicates the maximum number of rows into a drop down list.

Use the DropDownRows property to ensure the number of visible items into control's drop down window. The DropDownRows property has effect only if the control's EditType property is one of the followings: DropDown, DropDownList, CheckList, Font, PickEdit or ColorList.

property Editor.DropDownVisible as Boolean

Retrieves or sets a value that indicates whether the control's drop down window is shown or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the control's drop down window is visible or hidden.

The DropDownVisible property hides the drop down button. The property has effect only if the control's EditType property is one of the followings: DropDown, DropDownList, CheckList, Date, Color, Font, Picture, PickEdit or ColorList.

property Editor.EditText as String

Specifies the editor's caption.

Type	Description
String	A string expression that indicates the text inside an editor that contains an edit control.

Use the EditText property to specify the text inside the editor while the user edits the text. The property has effect only if the editor displays an edit control inside.

property Editor.EditType as EditTypeEnum

Retrieves or sets a value that indicates the type of the contained editor.

Type	Description
EditTypeEnum	An EditTypeEnum expression that specifies the type of the control.

The EditType property specifies the control's type. Use the AddItem method to add new items to the control's predefined list, if the editor is of drop down type. Use the HasCheckBox property to assign a check box to the control. Use the AddButton method to add new buttons to the editor. The Value property of the control determines the control's value, and it determines the caption that control displays based on the control's type. The EditTypeEnum type defines all types of supported editors and how the Value property is interpreted.

property Editor.Enabled as Boolean

Retrieves or sets a value that indicates whether the control is enabled or disabled.

Type	Description
Boolean	A boolean expression that indicates whether the control is enabled or disabled.

Use the ReadOnly property to make your editor read only. If the ReadOnly property is True, the Value property cannot be changed by the user, but the buttons are enabled. If the Enabled property is False, the Value property cannot be changed by the user, and the buttons are disabled.

property Editor.EventParam(Parameter as Long) as Variant

Retrieves or sets a value that indicates the current's event parameter.

Type	Description
Parameter as Long	A long expression that indicates the index of the parameter being requested ie 0 means the first parameter, 1 means the second, and so on. If -1 is used the EventParam property retrieves the number of parameters. Accessing an not-existing parameter produces an OLE error, such as invalid pointer (E_POINTER)
Variant	A VARIANT expression that specifies the parameter's value.

The EventParam method is provided to allow changing the event's parameters passed by reference, even if your environment does not support changing it (uniPaas 1.5 (formerly known as eDeveloper), DBase, and so on). For instance, Unipaas event-handling logic cannot update ActiveX control variables by updating the received arguments. The EventParam(0) retrieves the value of the first parameter of the event, while the EventParam(1) = 0, changes the value of the second parameter to 0 (the operation is successfully, only if the parameter is passed by reference). The EventParam(-1) retrieves the number of the parameters of the current event.

Let's take the event "event KeyDown (**KeyCode** as Integer, ByVal Shift as Integer)", where the KeyCode parameter is passed by reference. For instance, put the KeyCode parameter on 0, and the arrow keys are disabled while the control has the focus.

In most languages you will type something like:

```
Private Sub Control1_KeyDown(KeyCode As Integer, Shift As Integer)
   KeyCode = 0
End Sub
```

In case your environment does not support events with parameters by reference, you can use a code like follows:

```
Private Sub Control1_KeyDown(KeyCode As Integer, Shift As Integer)
   Control1.EventParam(0) = 0
End Sub
```

In other words, the EventParam property provides the parameters of the current event for reading or writing access, even if your environment does not allow changing parameters by

reference.

Calling the EventParam property outside of an event produces an OLE error, such as pointer invalid, as its scope was designed to be used only during events.

property Editor.FindItem (Value as Variant) as Variant

Finds an item given its value or caption.

Type	Description
Value as Variant	A long expression that indicates the value of the item being searched, a string expression that indicates the caption of the item being searched.
Variant	A long value that indicates the value of the item giving its caption, or a string expression that indicates the caption of the item giving its value.

The FindItem property searches for an item in the control's predefined list. Use the AddItem method to add new entries to the control's predefined list. Use the Value property to get the control's value. The ItemCount property counts the number of items in the control's predefined list.

The following sample displays the value of the item "CanShare":

```
Private Sub Form_Load()
   With Editor1
      .AddItem 1, "CanLink", 1
      .AddItem 2, "CanShare", 2
      .AddItem 4, "CanMove", 3
      .AddItem 8, "CanRe store", 4
      .EditType = EXEDITORSLibCtl.CheckList
      .Value = 1 + 4 ' CanLink + CanMove

       Debug.Print .FindItem("CanShare")

   End With
End Sub
```

The sample displays **2** as a result.

The following sample displays the caption of the item with the 2 value:

```
Private Sub Form_Load()
   With Editor1
      .AddItem 1, "CanLink", 1
```

```
    .AddItem 2, "CanShare", 2
    .AddItem 4, "CanMove", 3
    .AddItem 8, "CanRe store", 4
    .EditType = EXEDITORSLibCtl.CheckList
    .Value = 1 + 4 ' CanLink + CanMove

    Debug.Print .FindItem(2)

  End With
End Sub
```

The sample displays **"CanShare"** as a result.

property Editor.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object that indicates the control's font.

Use the Font property to change the editor's font.

property Editor.ForeColor as Color

Retrieves or sets the control's foreground color.

Type	Description
Color	A color expression that indicates the control's foreground color.

Use the ForeColor property to change the editor's foreground color. Use the BackColor property to change the editor's background color.

property Editor.HasCheckBox as Boolean

Retrieves or sets a value that indicates whether the control's check box is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the control's check box is visible or hidden.

Use the HasCheckBox property to associate a check box to your editor. Use the CheckState property to change the check box state. Use the PartialCheck property to allow two (unchecked, checked) or three (unchecked, checked, partial-checked) states to your check box. The CheckStateChanged event notifies your application that the control's check box state is changing.

The following sample assigns a check box to an editor of Date type:

```
Private Sub Form_Load()
   With Editor1

      .Value = Date
      .EditType = EXEDITORSLibCtl.Date
       .HasCheckBox = True
       .CheckState = Checked

   End With
End Sub
```

property Editor.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	The HTMLPicture specifies the picture being associated to a key. It can be one of the followings: • a string expression that indicates the path to the picture file, being loaded. • a string expression that indicates the base64 encoded string that holds a picture object, Use the eximages tool to save your picture as base64 encoded format. • A Picture object that indicates the picture being added or replaced. (A Picture object implements IPicture interface), If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added.

The HTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the <img> tags. By default, the HTMLPicture collection is empty. Use the HTMLPicture property to add new pictures to be used in HTML captions. For instance, the HTMLPicture("pic1") = "c:\winnt\zapotec.bmp", loads the zapotec picture and associates the pic1 key to it. Any "<img>pic1</img>" sequence in HTML captions, displays the pic1 picture. On return, the HTMLPicture property retrieves a Picture object (this implements the IPictureDisp interface).

property Editor.hWnd as Long

Retrieves the control's window handle.

Type	Description
Long	A long expression that indicates the control's window handle.

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

method Editor.Images (Handle as Variant)

Sets the control's handle image list.

Type	Description
Handle as Variant	The Handle parameter can be: • A string expression that specifies the ICO file to add. The ICO file format is an image file format for computer icons in Microsoft Windows. ICO files contain one or more small images at multiple sizes and color depths, such that they may be scaled appropriately. For instance, Images("c:\temp\copy.ico") method adds the sync.ico file to the control's Images collection *(string, loads the icon using its path)* • A string expression that indicates the BASE64 encoded string that holds the icons list. Use the Exontrol's ExImages tool to save/load your icons as BASE64 encoded format. In this case the string may begin with "gBJJ..." *(string, loads icons using base64 encoded string)* • A reference to a Microsoft ImageList control (mscomctl.ocx, MSComctlLib.ImageList type) that holds the icons to add *(object, loads icons from a Microsoft ImageList control)* • A reference to a Picture (IPictureDisp implementation) that holds the icon to add. For instance, the VB's LoadPicture (Function LoadPicture([FileName], [Size], [ColorDepth], [X], [Y]) As IPictureDisp) or LoadResPicture (Function LoadResPicture(id, restype As Integer) As IPictureDisp) returns a picture object (object, loads icon from a Picture object) • A long expression that identifies a handle to an Image List Control (the Handle should be of HIMAGELIST type). On 64-bit platforms, the Handle parameter must be a Variant of LongLong / LONG_PTR data type (signed 64-bit (8-byte) integers), saved under llVal field, as VT_I8 type. The LONGLONG / LONG_PTR is __int64, a 64-bit integer. For instance, in C++ you can use as Images(COleVariant((LONG_PTR)hImageList)) or Images(COleVariant((LONGLONG)hImageList)), where hImageList is of

HIMAGELIST type. The GetSafeHandle() method of the CImageList gets the HIMAGELIST handle (long, loads icon from HIMAGELIST type)

The user can add images at design time, by drag and drop files to control's image panel. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ReplaceIcon method to add, replace or remove icons in the control's Images collection at runtime. Use the ShowImageList property to hide the control's Images panel.

The following sample uses the Microsoft Image List control:

```
Editor1.Images ImageList1.hImageList
```

property Editor.ImageSize as Long

Retrieves or sets the size of icons the control displays.

Type	Description
Long	A long expression that defines the size of icons the control displays.

By default, the ImageSize property is 16 (pixels). The ImageSize property specifies the size of icons being loaded using the Images method. The control's Images collection is cleared if the ImageSize property is changed, so it is recommended to set the ImageSize property before calling the Images method. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. For instance, if the ICO file to load includes different types the one closest with the size specified by ImageSize property is loaded by Images method. The ImageSize property does NOT change the height for the control's font.

property Editor.ItemCaption (Index as Variant) as Variant

Gets the item's caption giving its index.

Type	Description
Index as Variant	A long expression that indicates the index of the item being retrieved.
Variant	A string expression that indicates the item's caption.

The ItemCaption property gets the caption of the item by index. Use the FindItem property to look for an item giving its value or its caption. The ItemCount property counts the number of items in the control's predefined list. The AddItem method adds new entries to the control's predefined list.

The following sample displays the items in the control's predefined list:

```
Private Sub Form_Load()
   With Editor1

      .AddItem 1, "CanLink", 1
      .AddItem 2, "CanShare", 2
      .AddItem 4, "CanMove", 3
      .AddItem 8, "CanRestore", 3
      .EditType = CheckList
      .Value = 1 + 4 ' CanLink + CanMove

       For i = 0 To .ItemCount - 1
          Debug.Print .ItemCaption(i)
       Next

   End With
End Sub
```

property Editor.ItemCount as Long

Counts the items in the collection.

Type	Description
Long	A long expression that indicates the number of items in the control's predefined list.

The ItemCount property counts the items in the control's predefined list. Use the AddItem method to add new entries to the control's predefined list. Use the RemoveItem property to remove an item from the control's predefined list. Use the FindItem property to look for an item. Use the ItemCaption property to get the caption of the item by specifying its index in the control's predefined list.

The following sample displays the items in the control's predefined list:

```
Private Sub Form_Load()
   With Editor1

      .AddItem 1, "CanLink", 1
      .AddItem 2, "CanShare", 2
      .AddItem 4, "CanMove", 3
      .AddItem 8, "CanRestore", 3
      .EditType = CheckList
      .Value = 1 + 4 ' CanLink + CanMove

      For i = 0 To .ItemCount - 1
         Debug.Print .ItemCaption(i)
      Next

   End With
End Sub
```

property Editor.Mask as String

Retrieves or sets a value that indicates the mask used by the editor.

Type	Description
String	A string expression that indicates the editor's mask.

Use the Mask property to filter characters during data input. Use the MaskChar property to change the masking character. If the Mask property is empty no filter is applied. The Mask property is composed by a combination of regular characters, literal escape characters, and masking characters. The Mask property can contain also alternative characters, or range rules. A literal escape character is preceded by a \ character, and it is used to display a character that is used in masking rules. Here's the list of all rules and masking characters:

Here's the list of all rules and masking characters.

- **#** (Digit), Masks a digit character, [0-9]
- **x** (Hexa Lower), Masks a lower case hexa character, [0-9],[a-f]
- **X** (Hexa Upper), Masks an upper case hexa character, [0-9],[A-F]
- **A** (AlphaNumeric), Masks a letter or a digit. [0-9], [a-z], [A-Z]
- **?** (Alphabetic), Masks a letter. [a-z],[A-Z]
- **<** (Alphabetic lower), Masks a lower case letter. [a-z]
- **>** (Alphabetic upper), Masks an upper case letter. [A-Z]
- ***** (Any), Masks any combination of characters.
- **** (Literal Escape), Displays any masking characters. The following combinations are valid: \#,\x,\X,\A,\?,\<,\>,\\,\{,\[
- **{**nMin**,**nMax**}** (Range), Masks a number in a range. The nMin and nMax values should be numbers. For instance the mask {0,255} will mask any number between 0 and 255.
- **[**...**]** (Alternative), Masks any characters that are contained in the [] brackets. For instance, the [abcA-C] mask any character: a,b,c,A,B,C

The following sample shows how to mask an IP address:

Editor1.Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"

The Mask property has effect for the following types: DropDown, Spin, Mask, Font, Button, PickEdit and LinkEdit.

property Editor.MaskChar as Long

Retrieves or sets a value that indicates the character used for masking.

Type	Description
Long	A long expression that indicates the ASCII code for the masking character.

The default masking character is '_' . Use the Mask property to filter characters during data input.

For instance, the following sample uses '0' for masking numbers into a masked number field:

```
With Editor1
   .EditType = EXEDITORSLibCtl.Mask
   .Mask = "####"
   .MaskChar = Asc("0")
   .Value = 10
End With
```

property Editor.Numeric as NumericEnum

Specifies whether the editor enables numeric values only.

Type	Description
NumericEnum	A NumericEnum expression that indicates whether integer or floating point numbers are allowed.

The Numeric property has effect only if the editor contains an edit box. Use the Numeric property to add intelligent input filtering for integer, or floating points numbers. Use the exSpinStep option to specify the proposed change when the user clicks a spin control, if the cell's editor is of SpinType type. Use the exEditDecimaSymbol option to specify the symbol being used by decimal value while editing a floating point number.

property Editor.Option(Name as EditorOptionEnum) as Variant

Specifies an option for the editor.

Type	Description
Name as EditorOptionEnum	An EditorOptionEnum expression that indicates the editor's option being changed.
Variant	A Variant expression that indicates the value for editor's option

The Option property of Editor object provides the ability to add scroll bars to a memo editor using the exMemoHScrollBar and exMemoVScrollBar options.

property Editor.PartialCheck as Boolean

Retrieves or sets a value that indicates whether the associated check box has two or three states.

Type	Description
Boolean	A boolean expression that indicates whether the associated check box has two or three states.

Specifies whether the editor's check box has two (unchecked, checked) or three (unchecked, checked, partial-checked) states. Use the HasCheckBox property to assign a check box to the control. Use the CheckState property to change the state of the control's check box.

The following sample assigns a check box to a Mask editor:

```
With Editor1
   .EditType = EXEDITORSLibCtl.Mask
   .Mask = "####"
   .MaskChar = Asc("0")
   .HasCheckBox = True
   .PartialCheck = True
   .Value = 10
End With
```

property Editor.PopupAppearance as InplaceAppearanceEnum

Retrieves or sets a value that indicates the popup window's appearance.

Type	Description
InplaceAppearanceEnum	An InplaceAppearanceEnum value that indicates the drop down window's appearance

Use the PopupAppearance property to change the popup's appearance. Use the Appearance to change the editor's appearance.

property Editor.ReadOnly as Boolean

Retrieves or sets a value that indicates whether the control is read-only.

Type	Description
Boolean	A boolean expression that indicates whether the editor is read only.

Use the ReadOnly property to make your editor read only. If the ReadOnly is True, the editor's Value cannot be changed by the user. For instance, if the editor is of one of the drop down types, the drop down window can be visible, but you cannot change the current value. Use the DropDownVisible property to hide the drop down button. Use the Enabled property to disable the editor.

method Editor.Refresh ()

Refreshes the control.

Type	Description

Use the Refresh method only if you need to force updating the control.

method Editor.RemoveButton (Key as Variant)

Removes a button given its key.

Type	Description
Key as Variant	A string expression that indicates the key of the button being removed.

Use the ClearButtons method to clear the buttons collection. Use the ButtonWidth property to hide all buttons. Use the AddButton method to adds new buttons to the control.

method Editor.RemoveItem (Value as Long)

Removes the item from editor's list given its value.

Type	Description
Value as Long	A long expression that indicates the value for the item being removed.

The RemoveItem method removes an item giving its value. Use the AddItem method to add new entries to the control's predefined list. Use the FindItem property to retrieve the value of the item giving its caption. Use the ClearItems method to remove all items.

method Editor.ReplaceIcon ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the control's image list.

Type	Description
Icon as Variant	A Variant expression that specifies the icon to add or insert, as one of the following options: • a long expression that specifies the handle of the icon (HICON) • a string expression that indicates the path to the picture file • a string expression that defines the picture's content encoded as BASE64 strings using the eXImages tool • a Picture reference, which is an object that holds image data. It is often used in controls like PictureBox, Image, or in custom controls (e.g., IPicture, IPictureDisp) If the Icon parameter is 0, it specifies that the icon at the given Index is removed. Furthermore, setting the Index parameter to -1 removes all icons. By default, if the Icon parameter is not specified or is missing, a value of 0 is used.
Index as Variant	A long expression that defines the index of the icon to insert or remove, as follows: • A zero or positive value specifies the index of the icon to insert (when Icon is non-zero) or to remove (when the Icon parameter is zero) • A negative value clears all icons when the Icon parameter is zero By default, if the Index parameter is not specified or is missing, a value of -1 is used.

Return	Description
Long	A long expression that indicates the index of the icon in the images collection

Use the ReplaceIcon property to add, remove or replace an icon in the control's images

collection. Also, the ReplaceIcon property can clear the images collection. Use the Images method to attach an image list to the control.

The following sample shows how to add a new icon to control's images list:

```
 i = Editor1.ReplaceIcon( LoadPicture("d:\icons\help.ico").Handle), where i is the index to insert the icon
```

The following sample shows how to replace an icon into control's images list::

```
 i = Editor1.ReplaceIcon( LoadPicture("d:\icons\help.ico").Handle, 0), in this case the i is zero, because the first icon was replaced.
```

The following sample shows how to remove an icon from control's images list:

```
 Editor1.ReplaceIcon 0,  i, in this case the i is the index of the icon to remove
```

The following sample shows how to clear the control's icons collection:

```
 Editor1.ReplaceIcon 0,  -1
```

property Editor.SelBackColor as Color

Retrieves or sets a value that indicates the selection background color.

Type	Description
Color	A color expression that indicates the background color of the selection in the control.

Use the SelBackColor and SelForeColor properties to change the color for the selection in the control.

property Editor.SelForeColor as Color

Retrieves or sets a value that indicates the selection foreground color.

Type	Description
Color	A color expression that indicates the foreground color of the selection in the control.

Use the SelBackColor and SelForeColor properties to change the color for the selection in the control.

property Editor.ShowImageList as Boolean

Specifies whether the control's image list window is visible or hidden.

Type	Description
Boolean	A boolean expression that specifies whether the control's image list window is visible or hidden.

Use the ShowImageList property to hide the control's Images panel. The Images panel of the control shows up only at design time. The ShowImageList property has effect only in design mode. Use the Images method to assign a list of icons to the control at run-time. Use the ReplaceIcon method to add, remove or replace icons at run-time.

method Editor.SortItems ([Ascending as Variant], [Reserved as Variant])

Sorts the list of items in the editor.

Type	Description
Ascending as Variant	A boolean expression that indicates the sort order of the items.
Reserved as Variant	For future use only.

Use the SortItems method to sort items in a drop down editor. Use the AddItem method to add new items to the control's predefined list.

property Editor.ToolTipDelay as Long

Specifies the time in ms that passes before the ToolTip appears.

Type	Description
Long	A long expression that specifies the time in ms that passes before the ToolTip appears.

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color.

property Editor.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

Type	Description
IFontDisp	A Font object being used to display the tooltip.

Use the ToolTipFont property to assign a font for the control's tooltip. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window. You can use the <*font*> HTML element, in the tooltip's description to assign a different font for portions of text.

property Editor.ToolTipPopDelay as Long

Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

Type	Description
Long	A long expression that specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to assign a font for the control's tooltip. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color.

property Editor.ToolTipWidth as Long

Specifies a value that indicates the width of the tooltip window, in pixels.

Type	Description
Long	A long expression that indicates the width of the tooltip window, in pixels.

Use the ToolTipWidth property to change the tooltip window width. The height of the tooltip window is automatically computed based on tooltip's description. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipFont property to assign a font for the control's tooltip. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color.

property Editor.UseVisualTheme as UIVisualThemeEnum

Specifies whether the control uses the current visual theme to display certain UI parts.

Type	Description
UIVisualThemeEnum	An UIVisualThemeEnum expression that specifies which UI parts of the control are shown using the current visual theme.

By default, the UseVisualTheme property is exDefaultVisualTheme, which means that all known UI parts are shown as in the current theme. The UseVisualTheme property may specify the UI parts that you need to enable or disable the current visual theme. The UI Parts are like check-boxes, buttons and so on. The UseVisualTheme property has effect only a current theme is selected for your desktop. The UseVisualTheme property. Use the Appearance property of the control to provide your own visual appearance using the EBN files.

property Editor.Value as Variant

Retrieves or sets the control's value

Type	Description
Variant	A Variant value that indicates the editor's value.

The Value property specifies the value of the editor. The control displays the value based on the type of the control. The type of the control is determined by the EditType property. The ValueChanged event occurs when user changes the control's data.

If the EditType property is None, the control is able to use built-in HTML format like follows:

- **bold**
- <u> underline </u>
- <s> ~~strikeout~~ </s>
- *italic*
- <fgcolor = FF0000> fgcolor </fgcolor>
- <bgcolor = FF0000> bgcolor </bgcolor>
-
 breaks a line.
- <solidline> draws a solid line
- <dotline> draws a dotted line
- <upline> draws the line to the top of the text line
- <r> aligns the rest of the text line to the right side.
- ****text displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.
- **&** glyph characters as **&** (&), **&lt** (<), **&gt** (>), **&qout** ("), **&#number**, For instance, the &#8364 displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display bold in HTML caption you can use bold

Newer HTML format supports subscript and superscript like follows:

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: *"Text with **<off** 6>subscript"* displays the text such as: Text with $_{\text{subscript}}$ The *"Text with **<off** -6>superscript"* displays the

text such as: Text with ^{subscript}

Also, newer HTML format supports decorative text like follows:

- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "*<**gra** FFFFFF;1;1>gradient-center</**gra**>*" generates the following picture:

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "*<**out** 000000> <fgcolor=FFFFFF>outlined</fgcolor></**out**>*" generates the following picture:

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "*<**sha**>shadow</**sha**>*" generates the following picture:

or "*<**sha** 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor> </**sha**>*" gets:

The following sample displays a simple caption:

```
With Editor1
   .EditType = EXEDITORSLibCtl.None
   .Value = "Just an <fgcolor=0000FF><b>HTML</b></fgcolor> caption."
End With
```

property Editor.Version as String

Retrieves the control's version.

Type	Description
String	A string expression that indicates the control's version.

The Version property gets the version of the control.

property Editor.VisualAppearance as Appearance

Retrieves the control's appearance.

Type	Description
Appearance	An Appearance object that holds a collection of skins.

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (*.ebn) assigned to a part of the control. By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part.

ExEditors events

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {ED8B26D2-1855-42D5-B622-FEE85395DFB3}. The object's program identifier is: "Exontrol.Editor". The /COM object module is: "ExEditors.dll"

The ExEditors control supports the following events:

Name	Description
ButtonClick	Occurs when the user clicks one of the control's buttons.
CheckStateChanged	Occurs when the control's check box state has been changed
Click	Occurs when the user presses and then releases the left mouse button over the control.
DblClick	Occurs when the user dblclk the left mouse button over the control.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
MouseDown	Occurs when the user presses the mouse button.
MouseMove	Occurs when the user moves the mouse.
MouseUp	Occurs when the user releases the mouse button.
ValueChanged	Occurs just before changing the control's Value.

event ButtonClick (Key as Variant)

Occurs when the user clicks one of the control's buttons.

Type	Description
Key as Variant	A string expression that indicates the key of the button clicked.

The ButtonClick event notifies your application that the user clicks a button. The AddButton method adds a new button to the editor. Also, the ButtonClick event is fired when user clicks the drop down button of an editor of one of the following types: DropDown, DropDownList, CheckList, Date, Color, Font, Picture, PickEdit and Button . In this case, the Key parameter is an empty string.

Syntax for ButtonClick event, **/NET** version, on:

C#

```
private void ButtonClick(object sender,object Key)
{
}
```

VB

```
Private Sub ButtonClick(ByVal sender As System.Object,ByVal Key As Object)
Handles ButtonClick
End Sub
```

Syntax for ButtonClick event, **/COM** version, on:

C#

```
private void ButtonClick(object sender,
AxEXEDITORSLib._IEditorEvents_ButtonClickEvent e)
{
}
```

C++

```
void OnButtonClick(VARIANT Key)
{
}
```

C++ Builder

```
void __fastcall ButtonClick(TObject *Sender,Variant Key)
{
}
```

Delphi

```
procedure ButtonClick(ASender: TObject; Key : OleVariant);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure ButtonClick(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_ButtonClickEvent);
begin
end;
```

Powe…

```
begin event ButtonClick(any Key)
end event ButtonClick
```

VB.NET

```
Private Sub ButtonClick(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_ButtonClickEvent) Handles ButtonClick
End Sub
```

VB6

```
Private Sub ButtonClick(ByVal Key As Variant)
End Sub
```

VBA

```
Private Sub ButtonClick(ByVal Key As Variant)
End Sub
```

VFP

```
LPARAMETERS Key

```

Xbas…

```
PROCEDURE OnButtonClick(oEditor,Key)
RETURN
```

Syntax for ButtonClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="ButtonClick(Key)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ButtonClick(Key)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComButtonClick Variant llKey
   Forward Send OnComButtonClick llKey
End_Procedure
```

Visual Objects

```
METHOD OCX_ButtonClick(Key) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ButtonClick(COMVariant _Key)
{
}
```

XBasic

```
function ButtonClick as v (Key as A)
end function
```

dBASE

```
function nativeObject_ButtonClick(Key)
return
```

The following sample displays the key of the button that user clicked:

```
Private Sub Editor1_ButtonClick(ByVal Key As Variant)
    Debug.Print "The user clicks the " & Iif(Key = "", "drop down", "'" & Key & "'")
& " button"
End Sub

Private Sub Form_Load()
   With Editor1

      .EditType = DropDownList
      .AddItem 1, "One"
      .AddItem 2, "Two"

      .AddButton "KeyA", 1

      .Value = 1
   End With
```

```
End Sub
```

event CheckStateChanged ()

Occurs when the control's check box state has been changed

Type	Description

The CheckStateChanged event notifies your application that user changes the checkbox state. Use the HasCheckBox property to associate a checkbox to the editor. Use the CheckState property to retrieve the state of the editor's checkbox. The PartialCheck property specifies whether the checkbox of the editor allows two (unchecked, checked) or three states (unchecked, checked and partial-checked).

Syntax for CheckStateChanged event, **/NET** version, on:

C#

```
private void CheckStateChanged(object sender)
{
}
```

VB

```
Private Sub CheckStateChanged(ByVal sender As System.Object) Handles
CheckStateChanged
End Sub
```

Syntax for CheckStateChanged event, **/COM** version, on:

C#

```
private void CheckStateChanged(object sender, EventArgs e)
{
}
```

C++

```
void OnCheckStateChanged()
{
}
```

C++ Builder

```
void __fastcall CheckStateChanged(TObject *Sender)
{
}
```

Delphi

```
procedure CheckStateChanged(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure CheckStateChanged(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event CheckStateChanged()
end event CheckStateChanged
```

VB.NET

```
Private Sub CheckStateChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CheckStateChanged
End Sub
```

VB6

```
Private Sub CheckStateChanged()
End Sub
```

VBA

```
Private Sub CheckStateChanged()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnCheckStateChanged(oEditor)
RETURN
```

Syntax for CheckStateChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="CheckStateChanged()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function CheckStateChanged()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComCheckStateChanged
    Forward Send OnComCheckStateChanged
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_CheckStateChanged() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CheckStateChanged()
{
}
```

XBasic

```
function CheckStateChanged as v ()
end function
```

dBASE

```
function nativeObject_CheckStateChanged()
return
```

The following sample associates a check box to the editor and prints the state of the check box when user clicks the editor's checkbox area.

```
Private Sub Editor1_CheckStateChanged()
    With Editor1
        Debug.Print "The user changes the state of the editor's checkbox to " &
.CheckState
    End With
End Sub

Private Sub Form_Load()
   With Editor1

      .EditType = DropDownList
      .AddItem 1, "One"
      .AddItem 2, "Two"

      .HasCheckBox = True

      .Value = 1
   End With
End Sub
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the control.

Type	Description

The Click event is fired when the user releases the left mouse button over the control. Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers

Syntax for Click event, **/NET** version, on:

C#

```
private void Click(object sender)
{
}
```

VB

```
Private Sub Click(ByVal sender As System.Object) Handles Click
End Sub
```

Syntax for Click event, **/COM** version, on:

C#

```
private void ClickEvent(object sender, EventArgs e)
{
}
```

C++

```
void OnClick()
{
}
```

C++ Builder

```
void __fastcall Click(TObject *Sender)
{
}
```

Delphi

```
procedure Click(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure ClickEvent(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event Click()
end event Click
```

VB.NET

```
Private Sub ClickEvent(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClickEvent
End Sub
```

VB6

```
Private Sub Click()
End Sub
```

VBA

```
Private Sub Click()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnClick(oEditor)
RETURN
```

Syntax for Click event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Click()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Click()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComClick
    Forward Send OnComClick
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_Click() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Click()
{
}
```

XBasic

```
function Click as v ()
end function
```

dBASE

```
function nativeObject_Click()
return
```

event DblClick (Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user double clicks the left mouse button over the control.

Type	Description
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The DblClick event is fired when the user dbl clicks on the control. Use the DblClick event to notify your application that a cell has been double-clicked.

Syntax for DblClick event, **/NET** version, on:

C#
```
private void DblClick(object sender,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub DblClick(ByVal sender As System.Object,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles DblClick
End Sub
```

Syntax for DblClick event, **/COM** version, on:

C#
```
private void DblClick(object sender, AxEXEDITORSLib._IEditorEvents_DblClickEvent e)
{
}
```

C++
```
void OnDblClick(short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall DblClick(TObject *Sender,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure DblClick(ASender: TObject; Shift : Smallint;X : Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure DblClick(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_DblClickEvent);
begin
end;
```

Powe…

```
begin event DblClick(integer Shift,long X,long Y)
end event DblClick
```

VB.NET

```
Private Sub DblClick(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_DblClickEvent) Handles DblClick
End Sub
```

VB6

```
Private Sub DblClick(Shift As Integer,X As Single,Y As Single)
End Sub
```

VBA

```
Private Sub DblClick(ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Shift,X,Y
```

Xbas…

```
PROCEDURE OnDblClick(oEditor,Shift,X,Y)
RETURN
```

Syntax for DblClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="DblClick(Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function DblClick(Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComDblClick Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY
    Forward Send OnComDblClick llShift llX llY
End_Procedure
```

Visual Objects

```
METHOD OCX_DblClick(Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_DblClick(int _Shift,int _X,int _Y)
{
}
```

XBasic

```
function DblClick as v (Shift as N,X as OLE::Exontrol.Editor.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Editor.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_DblClick(Shift,X,Y)
return
```

event KeyDown (ByRef KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	(By Reference) An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
```

In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

Syntax for KeyDown event, **/NET** version, on:

C#

```
private void KeyDown(object sender,ref short KeyCode,short Shift)
{
}
```

VB

```
Private Sub KeyDown(ByVal sender As System.Object,ByRef KeyCode As
Short,ByVal Shift As Short) Handles KeyDown
End Sub
```

Syntax for KeyDown event, **/COM** version, on:

C#

```
private void KeyDownEvent(object sender,
AxEXEDITORSLib._IEditorEvents_KeyDownEvent e)
{
}
```

C++

```
void OnKeyDown(short FAR* KeyCode,short Shift)
{
}
```

C++ Builder

```
void __fastcall KeyDown(TObject *Sender,short * KeyCode,short Shift)
{
}
```

Delphi

```
procedure KeyDown(ASender: TObject; var KeyCode : Smallint;Shift : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure KeyDownEvent(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_KeyDownEvent);
begin
end;
```

Powe…

```
begin event KeyDown(integer KeyCode,integer Shift)
end event KeyDown
```

VB.NET

```
Private Sub KeyDownEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_KeyDownEvent) Handles KeyDownEvent
End Sub
```

VB6

```
Private Sub KeyDown(KeyCode As Integer,Shift As Integer)
End Sub
```

VBA

```
Private Sub KeyDown(KeyCode As Integer,ByVal Shift As Integer)
End Sub
```

VFP

```
LPARAMETERS KeyCode,Shift
```

Xbas…
```
PROCEDURE OnKeyDown(oEditor,KeyCode,Shift)
RETURN
```

Syntax for KeyDown event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="KeyDown(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function KeyDown(KeyCode,Shift)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComKeyDown Short llKeyCode Short llShift
    Forward Send OnComKeyDown llKeyCode llShift
End_Procedure
```

Visual Objects
```
METHOD OCX_KeyDown(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_KeyDown(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic
```
function KeyDown as v (KeyCode as N,Shift as N)
end function
```

dBASE
```
function nativeObject_KeyDown(KeyCode,Shift)
return
```

event KeyPress (ByRef KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	(By Reference) An integer that returns a standard numeric ANSI keycode

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

Syntax for KeyPress event, **/NET** version, on:

C#

```
private void KeyPress(object sender,ref short KeyAscii)
{
}
```

VB

```
Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

Syntax for KeyPress event, **/COM** version, on:

C#

```
private void KeyPressEvent(object sender,
AxEXEDITORSLib._IEditorEvents_KeyPressEvent e)
{
}
```

C++

```
void OnKeyPress(short FAR* KeyAscii)
{
}
```

C++ Builder

```
void __fastcall KeyPress(TObject *Sender,short * KeyAscii)
{
}
```

Delphi

```
procedure KeyPress(ASender: TObject; var KeyAscii : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure KeyPressEvent(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_KeyPressEvent);
begin
end;
```

Powe…

```
begin event KeyPress(integer KeyAscii)
end event KeyPress
```

VB.NET

```
Private Sub KeyPressEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_KeyPressEvent) Handles KeyPressEvent
End Sub
```

VB6

```
Private Sub KeyPress(KeyAscii As Integer)
End Sub
```

VBA

```
Private Sub KeyPress(KeyAscii As Integer)
End Sub
```

VFP

```
LPARAMETERS KeyAscii

```

Xbas…

```
PROCEDURE OnKeyPress(oEditor,KeyAscii)
RETURN
```

Syntax for KeyPress event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyPress(KeyAscii)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyPress Short llKeyAscii
   Forward Send OnComKeyPress llKeyAscii
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyPress(KeyAscii) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyPress(COMVariant /*short*/ _KeyAscii)
{
}
```

XBasic

```
function KeyPress as v (KeyAscii as N)
end function
```

dBASE

```
function nativeObject_KeyPress(KeyAscii)
return
```

event KeyUp (ByRef KeyCode as Integer, Shift as Integer)

Occurs when the user releases a key while an object has the focus.

Type	Description
KeyCode as Integer	(By Reference) An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use the KeyUp event procedure to respond to the releasing of a key.

Syntax for KeyUp event, **/NET** version, on:

C#

```
private void KeyUp(object sender,ref short KeyCode,short Shift)
{
}
```

VB

```
Private Sub KeyUp(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal
Shift As Short) Handles KeyUp
End Sub
```

Syntax for KeyUp event, **/COM** version, on:

C#

```
private void KeyUpEvent(object sender,
AxEXEDITORSLib._IEditorEvents_KeyUpEvent e)
{
}
```

C++

```
void OnKeyUp(short FAR* KeyCode,short Shift)
{
}
```

C++ Builder

```
void __fastcall KeyUp(TObject *Sender,short * KeyCode,short Shift)
{
```

```
}
```

Delphi

```
procedure KeyUp(ASender: TObject; var KeyCode : Smallint;Shift : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure KeyUpEvent(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_KeyUpEvent);
begin
end;
```

Powe…

```
begin event KeyUp(integer KeyCode,integer Shift)
end event KeyUp
```

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_KeyUpEvent) Handles KeyUpEvent
End Sub
```

VB6

```
Private Sub KeyUp(KeyCode As Integer,Shift As Integer)
End Sub
```

VBA

```
Private Sub KeyUp(KeyCode As Integer,ByVal Shift As Integer)
End Sub
```

VFP

```
LPARAMETERS KeyCode,Shift

```

Xbas…

```
PROCEDURE OnKeyUp(oEditor,KeyCode,Shift)
RETURN
```

Syntax for KeyUp event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyUp(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyUp(KeyCode,Shift)
End Function
```

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyUp Short llKeyCode Short llShift
    Forward Send OnComKeyUp llKeyCode llShift
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyUp(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyUp(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic

```
function KeyUp as v (KeyCode as N,Shift as N)
end function
```

dBASE

```
function nativeObject_KeyUp(KeyCode,Shift)
return
```

event MouseDown (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user presses the mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Syntax for MouseDown event, **/NET** version, on:

C#

```
private void MouseDownEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB

```
Private Sub MouseDownEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseDownEvent
End Sub
```

Syntax for MouseDown event, **/COM** version, on:

C#

```
private void MouseDownEvent(object sender,
AxEXEDITORSLib._IEditorEvents_MouseDownEvent e)
{
}
```

C++

```
void OnMouseDown(short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall MouseDown(TObject *Sender,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure MouseDown(ASender: TObject; Button : Smallint;Shift : Smallint;X :
Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure MouseDownEvent(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_MouseDownEvent);
begin
end;
```

Powe…

```
begin event MouseDown(integer Button,integer Shift,long X,long Y)
end event MouseDown
```

VB.NET

```
Private Sub MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_MouseDownEvent) Handles MouseDownEvent
End Sub
```

VB6

```
Private Sub MouseDown(Button As Integer,Shift As Integer,X As Single,Y As Single)
End Sub
```

VBA

```
Private Sub MouseDown(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Button,Shift,X,Y

```

Xbas…

```
PROCEDURE OnMouseDown(oEditor,Button,Shift,X,Y)
RETURN
```

Syntax for MouseDown event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="MouseDown(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function MouseDown(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComMouseDown Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseDown llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_MouseDown(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_MouseDown(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function MouseDown as v (Button as N,Shift as N,X as
OLE::Exontrol.Editor.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Editor.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_MouseDown(Button,Shift,X,Y)
return
```

event MouseMove (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user moves the mouse.

Type	Description
Button as Integer	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

Syntax for MouseMove event, **/NET** version, on:

C#
```
private void MouseMoveEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub MouseMoveEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseMoveEvent
End Sub
```

Syntax for MouseMove event, **/COM** version, on:

C#
```
private void MouseMoveEvent(object sender,
AxEXEDITORSLib._IEditorEvents_MouseMoveEvent e)
{
}
```

C++
```
void OnMouseMove(short Button,short Shift,long X,long Y)
```

```
{
}
```

C++ Builder

```
void __fastcall MouseMove(TObject *Sender,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure MouseMove(ASender: TObject; Button : Smallint;Shift : Smallint;X :
Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure MouseMoveEvent(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_MouseMoveEvent);
begin
end;
```

Powe…

```
begin event MouseMove(integer Button,integer Shift,long X,long Y)
end event MouseMove
```

VB.NET

```
Private Sub MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_MouseMoveEvent) Handles MouseMoveEvent
End Sub
```

VB6

```
Private Sub MouseMove(Button As Integer,Shift As Integer,X As Single,Y As Single)
End Sub
```

VBA

```
Private Sub MouseMove(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Button,Shift,X,Y

```

Xbas…

```
PROCEDURE OnMouseMove(oEditor,Button,Shift,X,Y)
RETURN
```

Syntax for MouseMove event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="MouseMove(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function MouseMove(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComMouseMove Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseMove llButton llShift llX llY
End_Procedure
```

Visual Objects

```
METHOD OCX_MouseMove(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_MouseMove(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic

```
function MouseMove as v (Button as N,Shift as N,X as
OLE::Exontrol.Editor.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Editor.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_MouseMove(Button,Shift,X,Y)
return
```

event MouseUp (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user releases the mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Syntax for MouseUp event, **/NET** version, on:

C#

```
private void MouseUpEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB

```
Private Sub MouseUpEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseUpEvent
End Sub
```

Syntax for MouseUp event, **/COM** version, on:

C#

```
private void MouseUpEvent(object sender,
AxEXEDITORSLib._IEditorEvents_MouseUpEvent e)
{
```

```
}
```

C++

```
void OnMouseUp(short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall MouseUp(TObject *Sender,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure MouseUp(ASender: TObject; Button : Smallint;Shift : Smallint;X :
Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure MouseUpEvent(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_MouseUpEvent);
begin
end;
```

Powe…

```
begin event MouseUp(integer Button,integer Shift,long X,long Y)
end event MouseUp
```

VB.NET

```
Private Sub MouseUpEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_MouseUpEvent) Handles MouseUpEvent
End Sub
```

VB6

```
Private Sub MouseUp(Button As Integer,Shift As Integer,X As Single,Y As Single)
End Sub
```

VBA

```
Private Sub MouseUp(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Button,Shift,X,Y

```

Xbas…

```
PROCEDURE OnMouseUp(oEditor,Button,Shift,X,Y)
```

```
RETURN
```

Syntax for MouseUp event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="MouseUp(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function MouseUp(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComMouseUp Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseUp llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_MouseUp(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_MouseUp(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function MouseUp as v (Button as N,Shift as N,X as
OLE::Exontrol.Editor.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Editor.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_MouseUp(Button,Shift,X,Y)
return
```

event ValueChanged (ByRef NewValue as Variant)

Occurs just before changing the control's Value.

Type	Description
NewValue as Variant	(By Reference) A Variant expression that holds the new value for the editor.

The ValueChanged event is fired just before changing the editor's Value property. The ValueChanged event notifies your application that the value of the editor is changing. The FindItem property gets the caption associated to a value if the editor contains a predefined list (DropDownList).

Syntax for ValueChanged event, **/NET** version, on:

C#

```
private void ValueChanged(object sender,ref object NewValue)
{
}
```

VB

```
Private Sub ValueChanged(ByVal sender As System.Object,ByRef NewValue As
Object) Handles ValueChanged
End Sub
```

Syntax for ValueChanged event, **/COM** version, on:

C#

```
private void ValueChanged(object sender,
AxEXEDITORSLib._IEditorEvents_ValueChangedEvent e)
{
}
```

C++

```
void OnValueChanged(VARIANT FAR* NewValue)
{
}
```

C++ Builder

```
void __fastcall ValueChanged(TObject *Sender,Variant * NewValue)
{
}
```

Delphi

```
procedure ValueChanged(ASender: TObject; var NewValue : OleVariant);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure ValueChanged(sender: System.Object; e:
AxEXEDITORSLib._IEditorEvents_ValueChangedEvent);
begin
end;
```

Powe…

```
begin event ValueChanged(any NewValue)
end event ValueChanged
```

VB.NET

```
Private Sub ValueChanged(ByVal sender As System.Object, ByVal e As
AxEXEDITORSLib._IEditorEvents_ValueChangedEvent) Handles ValueChanged
End Sub
```

VB6

```
Private Sub ValueChanged(NewValue As Variant)
End Sub
```

VBA

```
Private Sub ValueChanged(NewValue As Variant)
End Sub
```

VFP

```
LPARAMETERS NewValue

```

Xbas…

```
PROCEDURE OnValueChanged(oEditor,NewValue)
RETURN
```

Syntax for ValueChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="ValueChanged(NewValue)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ValueChanged(NewValue)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComValueChanged Variant llNewValue
    Forward Send OnComValueChanged llNewValue
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_ValueChanged(NewValue) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ValueChanged(COMVariant /*variant*/ _NewValue)
{
}
```

XBasic

```
function ValueChanged as v (NewValue as A)
end function
```

dBASE

```
function nativeObject_ValueChanged(NewValue)
return
```

The following sample shows how to restore the old value after user changes the editor's data:

```
Option Explicit
Dim iChanging As Long

Private Sub Editor1_ValueChanged(NewValue As Variant)
    If (iChanging = 0) Then
        With Editor1
            NewValue = .Value
        End With
    End If
End Sub

Private Sub Form_Load()
    iChanging = 0
    With Editor1

        .EditType = DropDownList
        .AddItem 1, "One"
        .AddItem 2, "Two"
```

```
        iChanging = iChanging + 1
        .Value = 1
        iChanging = iChanging - 1
    End With
End Sub
```

The following sample displays the newly value:

```
Option Explicit
Dim iChanging As Long

Private Sub Editor1_ValueChanged(NewValue As Variant)
    With Editor1
        Debug.Print "The user changes the editor's value to " & NewValue
    End With
End Sub

Private Sub Form_Load()
    iChanging = 0
    With Editor1

        .EditType = DropDownList
        .AddItem 1, "One"
        .AddItem 2, "Two"

        iChanging = iChanging + 1
        .Value = 1
        iChanging = iChanging - 1
    End With
End Sub
```