

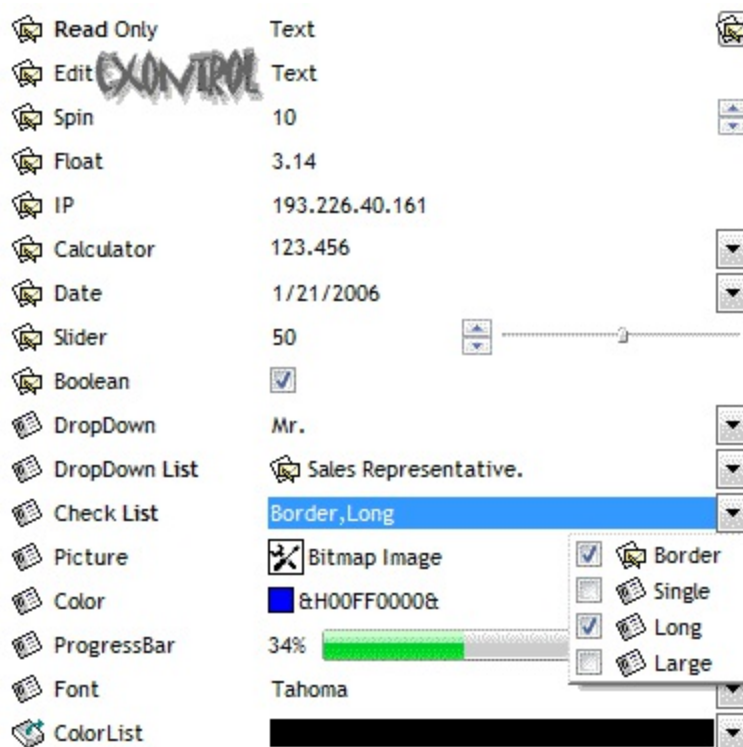


## ExRecord

Exontrol's new exRecord control is a container component that displays a set of editors added manually or bounded to a table in a database. The exRecord name comes from the record, that's a set of fields that contain related information, in database type systems. The exRecord significantly reduces development time of data components.

Features include:

- Skinnable Interface support ( ability to apply a skin to any background part )
- **ADO** and **DAO** data binding support
- **WYSWYG Template/Layout Editor** support
- It includes editors like: mask, date, drop down list, check box list, memo fields, spin, slider, OLE Object viewer, color, buttons and more.
- Ability to use custom **ActiveX** control as built-in **editors**
- **ActiveX hosting** (you can place any ActiveX component in any field of the control).
- Arranging fields from left to right, from top to bottom or custom layout as well.
- Ability to load icons and pictures from BASE64 encoded strings.
- Multi-lines HTML tooltip support



Ž ExRecord is a trademark of Exontrol. All Rights Reserved.

# 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](mailto: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 object's alignment.

Name	Value	Description
LeftAlignment	0	The object is left aligned.
CenterAlignment	1	The object is centered.
RightAlignment	2	The object is right aligned.

# constants AppearanceEnum

Specifies the object's appearance.

Name	Value	Description
None2	0	The source has no borders.
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	4	Specifies the visual appearance for the drop down button, when it is up.
exDropDownButtonDown	5	Specifies the visual appearance for the drop down button, when it is down.
exButtonUp	6	Specifies the visual appearance for the button inside the editor, when it is up.
exButtonDown	7	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.
exSliderRange	15	Specifies the visual appearance for the slider's bar.
exSliderThumb	16	Specifies the visual appearance for the thumb of the slider.
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.
exCheckBoxState0	70	Specifies the visual appearance for the check box in 0 state (unchecked).
exCheckBoxState1	71	Specifies the visual appearance for the check box in 1 state (checked).
exCheckBoxState2	72	Specifies the visual appearance for the check box in 2 state (partial, not used).

# constants CheckStateEnum

Specifies the check-box's states. Use the [CheckImage](#) property to assign new icons for check-box states. Use the [CheckValueType](#) editor to assign a check box to a field.

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

# constants EditorOptionEnum

Specifies different options for a built-in editor. The [Option](#) property specifies the editor's options.

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 )
exMemoAutoSize	3	Specifies whether the MemoType editor is resized when user alters the text. By default, the Editor.Option( exMemoAutoSize ) is True. ( boolean expression )
exColorListShowName	4	Specifies whether a ColorListType editor displays the name of the color. By default, the Editor.Option( exColorListShowName ) is False. ( boolean expression )
exColorShowPalette	5	Specifies whether the ColorList editor displays the palette colors list. By default, the Editor.Option( exColorShowPalette ) is True. ( boolean expression )
exColorShowSystem	6	Specifies whether the ColorType editor shows the system colors list. By default, the Editor.Option( exColorShowSystem ) is True. ( boolean expression )
exMemoDropDownWidth	7	Specifies the width for a MemoDropDownType editor. ( long expression )
exMemoDropDownHeight	8	Specifies the height for a MemoDropDownType editor. ( long expression )
exMemoDropDownAcceptReturn	9	Specifies whether the Return key is used to add new lines into a MemoDropDownType editor. (boolean expression)
exEditRight	10	Right-aligns text in a single-line or multiline edit control. (boolean expression)



exProgressBarBackColor	11	Specifies the background color for a progress bar editor. (color expression)
exProgressBarAlignment	12	Specifies the alignment of the caption inside of a progress bar editor. ( <a href="#">AlignmentEnum</a> expression )
exProgressBarMarkTicker	13	Retrieves or sets a value that indicates whether the ticker of a progress bar editor is visible or hidden. (boolean expression)
exDateAllowNullDate	14	Allows you to specify an empty date to a DateType editor. (boolean expression)
exCheckValue0	15	Specifies the check box state being displayed for unchecked state. (long expression, valid values are 0, 1 or 2)
exCheckValue1	16	Specifies the check box state being displayed for checked state. (long expression, valid values are 0, 1 or 2)
exCheckValue2	17	Specifies the check box state being displayed for partial checked state. (long expression, valid values are 0, 1 or 2)
exEditPassword	18	Specifies a value that indicates whether an edit control displays all characters as an asterisk (*) as they are typed ( passwords ). (boolean expression)
exEditPasswordChar	19	Specifies a value that indicates the password character. (character expression)
exLeftArrow	20	(VK_LEFT) Specifies whether the left arrow key is handled by the control or by the current editor. By default, the Option(exLeftArrow) property is True. Use the exLeftArrow option to disable focusing a new editor if the user presses the left arrow key while editing. The option is valid for all editors. (boolean expression)
exRightArrow	21	(VK_RIGHT) Specifies whether the right arrow key is handled by the control or by the current editor. By default, the Option(exRightArrow) property is True. Use the exRightArrow option to disable focusing a new editor if the user presses the right arrow key while editing. The option is valid for all editors. (boolean expression)
		(VK_UP) Specifies whether the up arrow key is handled by the control or by the current editor. By

exUpArrow	22	default, the Option(exUpArrow) property is True. Use the exUpArrow option to disable focusing a new editor if the user presses the up arrow key while editing. The option is valid for all editors. (boolean expression)
exDownArrow	23	(VK_DOWN) Specifies whether the down arrow key is handled by the control or by the current editor. By default, the Option(exDownArrow) property is True. Use the exDownArrow option to disable focusing a new editor if the user presses the down arrow key while editing. The option is valid for all editors.
exHomeKey	24	(VK_HOME) Specifies whether the home key is handled by the control or by the current editor. By default, the Option(exHomeKey) property is True. Use the exHomeKey option to disable focusing a new editor if the user presses the home key while editing. The option is valid for all editors. (boolean expression)
exEndKey	25	(VK_END) Specifies whether the end key is handled by the control or by the current editor. By default, the Option(exEndKey) property is True. Use the exEndKey option to disable focusing a new editor if the user presses the end key while editing. The option is valid for all editors. (boolean expression)
exPageUpKey	26	(VK_PRIOR) Specifies whether the page up key is handled by the control or by the current editor. By default, the Option(exPageUpKey) property is True. Use the exPageUpKey option to disable focusing a new editor if the user presses the page up key while editing. The option is valid for all editors. (boolean expression)
exPageDownKey	27	(VK_NEXT) Specifies whether the page down key is handled by the control or by the current editor. By default, the Option(exPageDownKey) property is True. Use the exPageDownKey option to disable focusing a new editor if the user presses the page down key while editing. The option is valid for all editors. (boolean expression)
		Displays the predefined icon in the control's editor, if the user selects an item from a drop down editor.

exDropDownImage	28	By default, the exDropDownImage property is True. The option is valid for DropDownListType, PickEdit and ColorListType editors. (boolean expression)
exDateTodayCaption	29	Specifies the caption for the 'Today' button in a DateType editor. By default, the Editor.Option(exDateTodayCaption) is "Today". (string expression)
exDateMonths	30	Specifies the name for months to be displayed in a DateType editor. The list of months should be delimited by spaces. By default, the Editor.Option(exDateMonths) = "January February March April May June July August September October November December". (string expression)
exDateWeekDays	31	Specifies the shortcut for the weekdays to be displayed in a DateType editor. The list of shortcut for the weekdays should be separated by spaces. By default, the Editor.Option(exDateWeekDays) = "S M T W T F S". The first shortcut in the list indicates the shortcut for the Sunday, the second shortcut indicates the shortcut for Monday, and so on. (string expression)
exDateFirstWeekDay	32	Specifies the first day of the week in a DateType editor. By default, the Editor.Option(exDateFirstWeekDay) = 0. The valid values for the Editor.Option(exDateFirstWeekDay) property are like follows: 0 - Sunday, 1 - Monday, 2 - Tuesday, 3 - Wednesday, 4 - Thursday, 5 - Friday and 6 - Saturday. (long expression, valid values are 0 to 6)
exDateShowTodayButton	33	Specifies whether the 'Today' button is visible or hidden in a DateType editor. By default, the Editor.Option(exDateShowTodayButton) property is True. (boolean expression)
exDateMarkToday	34	Gets or sets a value that indicates whether the today date is marked in a DateType editor. By default, Editor.Option(exDateMarkToday) property is False. (boolean expression)
exDateShowScroll	35	Specifies whether the years scroll bar is visible or hidden in a DateType editor. By default, the Editor.Option(exDateShowScroll) property is

True. (boolean expression)

exEditLimitText

36

Limits the length of the text that the user may enter into an edit control. By default, the Editor.Option(exEditLimitText) is zero, and so no limit is applied to the edit control. (long expression)

exAutoDropDownList

37

The exAutoDropDownList has no effect Editor.Option( exAutoDropDownList) property is 0 ( default ). Automatically shows the drop down list when user starts typing characters into a DropDownList editor, if the Editor.Option(exAutoDropDownList) property is -1. If the Editor.Option( exAutoDropDownList) property is +1, the control selects a new item that matches typed characters without opening the drop down portion of the editor. (long expression, valid values are -1, 0 and +1)

exExpandOnSearch

38

Expands items while user types characters into a drop down editor. The exExpandOnSearch type has effect for drop down type editors. ( boolean expression )

exAutoSearch

39

Only for future use.

exSpinStep

40

Specifies the proposed change when user clicks a spin control. The exSpinStep should be a positive number, else clicking the spin has no effect. By default, the exSpinStep option is 1. Integer or floating points allowed as well. he exSliderTickFrequency property specifies the frequency to display ticks on a slider control. For instance, if the exSpinStep is 0.01, the proposed change when user clicks the spin is 0.01. If the exSpinStep property is 0, the spin control is hidden ( useful if you have a slider control ).

exSliderWidth

41

Specifies the width in pixels of the slider control. The exSliderWidth value could be 0, when the slider control is hidden, a positive value that indicates the width in pixels of the slider in the control, a negative number when its absolute value indicates the percent of the editor's size being used by the slider. For instance, Option(exSliderWidth) = 0, hides the slider, Option(exSliderWidth) = 100, shows a slider of 100 pixels width, Option(exSliderWidth) = -50,

uses half of the editor's client area to display a slider control. By default the Option(exSliderWidth) property is 64 pixels. Use the exSpinStep to hide the spin control. ( long expression )

exSliderStep	42	Specifies a value that represents the proposed change in the slider control's position. ( double expression , by default it is 1 )
exSliderMin	43	Specifies the slider's minimum value. ( double expression, by default it is 0 )
exSliderMax	44	Specifies the slider's maximum value. ( double expression, by default it is 100 )
exKeepSelBackColor	45	Keeps the selection background color while the editor is visible. The exKeepSelBackColor option is valid for all editors. By default, the Option(exKeepSelBackColor) property is False. Use the exKeepSelBackColor to let the editor to display the control's selection background color when it is visible. ( boolean expression )
exEditDecimalSymbol	46	Specifies the symbol that indicates the decimal values while editing a floating point number. By default, the exEditDecimalSymbol value is the "Decimal symbol" settings as in the Regional Options, in your control panel. Use the exEditDecimaSymbol option to assign a different symbol for floating point numbers, when <a href="#">Numeric</a> property is exFloat. (long expression, that indicates the ASCII code for the character being used as decimal symbol.)
exDateWeeksHeader	47	Sets or gets a value that indicates whether the weeks header is visible or hidden in a DateType editor. By default, Editor.Option(exDateWeeksHeader) property is False. ( boolean expression ).
exEditSelStart	48	Sets the starting point of text selected, when an EditType editor is opened.
exEditSelLength	49	Sets the number of characters selected, when an EditType editor is opened.
exEditLockedBackColor	50	Specifies the background color for a locked edit control.

exEditLockedForeColor	51	Specifies the foreground color for a locked edit control.
exSliderTickFrequency	53	Gets or sets the interval between tick marks slider types. By default, the exSliderTickFrequency property is 0 which makes the slider to display no ticks. The exSliderTickFrequency property specifies the frequency to display ticks on a slider control. The exSliderStep proposed change in the slider control's position. The exSliderMin and exSliderMax determines the range of values for the slider control. The exSliderWidth option specifies the width of the slider within the cell. ( double expression, by default it is 0 )
exPickAllowEmpty	54	Specifies whether the editor of PickEditType supports empty value.
exDropDownBackColor	55	Specifies the drop down's background color.
exDropDownForeColor	56	Specifies the drop down's foreground color.
exDropDownColumnCaption	57	Specifies the HTML caption for each column within the drop down list, separated by Š character (vertical broken bar, ALT + 221).
exDropDownColumnWidth	58	Specifies the width for each column within the drop down list, separated by Š character (vertical broken bar, ALT + 221).
exDropDownColumnPosition	59	Specifies the position for each column within the drop down list, separated by Š character (vertical broken bar, ALT + 221).
exDropDownColumnAutoResize	60	Specifies whether the drop down list resizes automatically its visible columns to fit the drop down width.
exSliderTickStyle	63	Gets or sets the style to display the slider' ticks.
exCalcExecuteKeys	100	Specifies whether the calculator editor executes the keys while focused and the drop down portion is hidden. ( boolean expression, by default it is True ).
exCalcCannotDivideByZero	101	Specifies the message whether a division by zero occurs in a calendar editor. ( string expression, by default it is "Cannot divide by zero." ).
exCalcButtonWidth	102	Specifies the width in pixels of the buttons in the calculator editor. ( long expression, by default it is

24 ).

exCalcButtonHeight

103

Specifies the height in pixels of the buttons in the calculator editor. ( long expression, by default it is 24 ).

exCalcButtons

104

Specifies buttons in a calendar editor. The property specifies the buttons and the layout of the buttons in the control. A string expression that indicates the list of buttons being displayed. The rows are separated by chr(13)+chr(10) ( vbCrLf ) sequence, and the buttons inside the row are separated by ';' character. ( string expression )

exCalcPictureUp

105

Specifies the picture when the button is up in a drop down calendar editor. A Picture object that indicates the node's picture. ( A Picture object implements IPicture interface ), 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.

exCalcPictureDown

106

Specifies the picture when the button is down in a drop down calendar editor. A Picture object that indicates the node's picture. ( A Picture object implements IPicture interface ), 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.

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 ).

exEditAllowContextMenu

202

exEditAllowContextMenu. Specifies whether the editor displays the edit's default context menu when the user right clicks the field.

# constants EditTypeEnum

Use the [EditType](#) property to specify the type of the editor. Use the [Add](#) method to add a new editor to the control. Use the [AddItem](#), [InsertItem](#) method to add new items to a drop down list editor. Use the [AddButton](#) method to add the buttons to the editor. Use the [Option](#) property to assign different options for a given editor. The exRecord component supports the following type of editors:

Name	Value	Description
ReadOnly	0	The editor is not ediatble.
EditType	1	<p>The editor supports the following options:</p> <ul style="list-style-type: none"><li>• <b>exEditRight</b>, Right-aligns text in a single-line or multiline edit control.</li><li>• <b>exEditPassword</b>, Specifies a value that indicates whether an edit control displays all characters as an asterisk (*) as they are typed ( passwords ).</li><li>• <b>exEditPasswordChar</b>, Specifies a value that indicates the password character.</li></ul> <p>The <a href="#">Numeric</a> property specifies whether the editor enables numeric values only.</p>

It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window, but it accepts new values at runtime too. The DropDownType editor has associated a standard text edit field too. Use [AddItem](#) method to add predefined values to the drop down list. Use the [InsertItem](#) method to insert child items to the editor's predefined list. The [DropDownRows](#) property specifies the maximum number of visible rows into the drop-down list. The editor displays the [Value](#) value.



The following sample adds a DropDownType editor:

```
DropDownType      2      With Record1
                        .BeginUpdate
                        With .Add("DropDownType",
```



```
EXRECORDLibCtl.DropDownType)
```

```
.AddItem 0, "Single Bed", 1
```

```
.AddItem 1, "Double Bed", 2
```

```
.AddItem 2, "Apartment", 3
```

```
.AddItem 3, "Suite", 4
```

```
.AddItem 4, "Royal Suite", 5
```

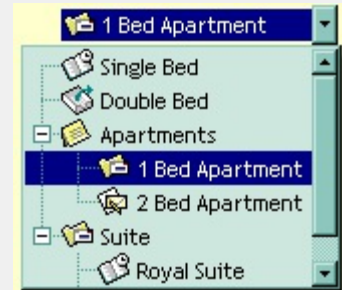
```
.Value = "Apartment"
```

```
End With
```

```
.EndUpdate
```

```
End With
```

It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. The DropDownListType editor has no standard edit field



associated. Use [AddItem](#)

method to add predefined values to the drop down list. Use [InsertItem](#) method to insert child predefined values to the drop down list. The [DropDownRows](#) property specifies the maximum number of visible rows into the drop-down list. The editor displays the caption of the item that matches the [Value](#) value. The item's icon is also displayed if it exists.

The following sample adds a DropDownListType editor:

DropDownListType

3

```
With Record1
```

```
.BeginUpdate
```

```
With .Add("DropDownType",  
EXRECORDLibCtl.DropDownListType)
```

```
.DropDownAutoWidth = False
```

```
.AddItem 0, "Single Bed", 1
```

```
.AddItem 1, "Double Bed", 2
```

```
.AddItem 2, "Apartments", 3
```


```
.InsertItem 3, "1 Bed Apartment", 4, 2
```

```
.InsertItem 4, "2 Bed Apartment", 5, 2
```

```

.AddItem 5, "Suite", 4
.InsertItem 6, "Royal Suite", 1, 5
.InsertItem 7, "Deluxe Suite", 2, 5
.ExpandAll
.Value = 3
End With
.EndUpdate
End With

```

The SpinType allows your users to view  and change numeric values using a familiar up/down button (spin control) combination. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is SpinType. Use the [exSpinStep](#) option to specify the proposed change when user clicks the spin button. The [Numeric](#) property specifies whether the editor enables numeric values only. Use the SliderType to specify minimum and maximum values for the spin.

The following sample adds a SpinType editor:

SpinType

4

```

With Record1
.BeginUpdate
With .Add("Spin", EXRECORDLibCtl.SpinType)
.Numeric = exFloat
.Option(exSpinStep) = 0.1
.Value = 3.14
End With
.EndUpdate
End With

```

MemoType

5

The MemoType is designed to provide a unique and intuitive interface, which you can implement within your application to assist users in working with textual information. If all information does not fit within the edit box, the window of the editor is enlarged. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is SpinType. You can use options like exMemoHScrollBar, exMemoVScrollBar and so on.

It provides an intuitive interface for your users to check values from pre-defined lists presented in a drop-down window. Each item has a check box associated. The editor displays the list of item captions, separated by comma, that is OR combination of the [Value](#) expression. The [AddItem](#) method adds new predefined values to the drop down portion of the editor. The [DropDownRows](#) property specifies the maximum number of visible rows into the drop-down list. Use the [CheckImage](#) property to assign a different icons for check box states.



The following sample adds a CheckListType editor:

CheckListType

6

```
With Record1
.BeginUpdate
  With .Add("CheckListType",
EXRECORDLibCtl.CheckListType)
    .AddItem 1, "Single Bed", 1
    .AddItem 2, "Double Bed", 2
    .AddItem 4, "Apartment", 3
    .AddItem 8, "Suite", 4
    .AddItem 16, "Royal Suite", 5
    .Value = 5
  End With
.EndUpdate
End With
```

The DateType editor is a date/calendar control ( not the Microsoft Calendar Control ). The dropdown calendar provides an efficient and appealing way to edit dates at runtime. The DateType editor has a standard edit control associated. The user can easy select a date by selecting a date from the drop down calendar, or



DateType

7

by typing directly the date. The editor displays the [Value](#) value as date. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is DateType.

The following sample adds a DateType editor:

```
With Record1
    .BeginUpdate
        With .Add("DateType",
EXRECORDLibCtl.DateType)
            .Value = Date
        End With
    .EndUpdate
End With
```

You can use the MaskType to enter any (074) 728 - 8| data that includes literals and requires a mask to filter the characters during data input. You can use this control to control the entry of many types of formatted information such as telephone numbers, social security numbers, IP addresses, license keys etc. The [Mask](#) property specifies the editor's mask. The [MaskChar](#) property specifies the masking character. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is MaskType. The Mask property can use one or more literals: #,x,X,A,? <,>,\*\,{nMin,nMax},[...]. The following sample shows how to mask the "MaskType" column for input telephone numbers.

MaskType

8

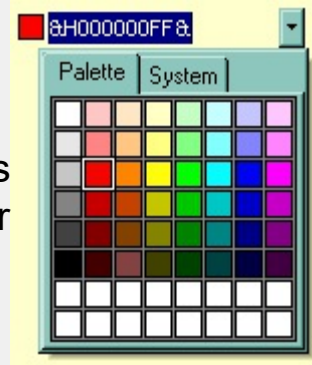
The following sample adds a MaskType editor:

```
With Record1
    .BeginUpdate
        With .Add("MaskType",
EXRECORDLibCtl.MaskType)
            .Mask = "(###) ### - ####"
            .Value = "(074) 728 - 2121"
        End With
    .EndUpdate
```

## ColorType

9

You can include a color selection control in your applications via the ColorType editor. Check the ColorListType also. The editor has a standard edit control and a color drop-down window. The color drop-down window contains two tabs that can be used to select colors, the "Palette" tab shows a grid of colors, while the "System" tab shows the current windows color constants. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is ColorType.



The following sample adds a ColorType editor:

```
With Record1
  .BeginUpdate
  With .Add("ColorType",
    EXRECORDLibCtl.ColorType)
    .Value = RGB(255, 0, 0)
  End With
  .EndUpdate
End With
```

## FontType

10

Provides an intuitive way for selecting fonts. The FontType editor contains a standard edit control and a font drop-down window. The font drop-down window contains a list with all system fonts. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is FontType. The [Value](#) property indicates the name of the font. The [DropDownRows](#) property specifies the maximum number of visible rows into the drop-down list.



The following sample adds a FontType editor:

```

With Record1
    .BeginUpdate
        With .Add("FontType",
EXRECORDLibCtl.FontType)
            .Value = "Tahoma"
        End With
    .EndUpdate
End With

```

The PictureType provides an elegant way for displaying the fields of OLE Object type and cells that have a reference to an IPicture interface. An OLE Object field can contain a picture, a Microsoft Clip Gallery, a package, a chart, PowerPoint



slide, a word document, a WordPad document, a wave file, and so on. In MS Access you can specify the field type to OLE Object. The [DropDownMinWidth](#) property specifies the minimum width for the drop-down window. The drop-down window is scaled based on the picture size. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is PictureType. If your control is bounded to a recordset, it automatically detects the OLE Object fields, so setting the editor's type to PictureType is not necessary.

The [Value](#) property specifies the source of the picture being displayed, and it can be


PictureType

11

- a Picture object ( A Picture object implements IPicture interface )
- a string expression that indicates the path to the picture file
- 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
- an array of bytes that holds an OLE object field

The following sample adds a PictureType editor:

```
With Record1
  .BeginUpdate
  With .Add("PictureType",
    EXRECORDLibCtl.PictureType)
    .Value = App.Path & "\xfmail.gif"
  End With
  .EndUpdate
End With
```


The ButtonType editor consists  into a standard edit field and a "...". The [ButtonClick](#) event is fired if the user has clicked the button. The [AddItem](#) or [InsertItem](#) method has no effect, if the EditType is ButtonType. The [ButtonWidth](#) property specifies the width of the button. The [Value](#) property specifies the editor's caption.

The following sample adds a ButtonType editor:

ButtonType

12

```
With Record1
  .BeginUpdate
  With .Add("ButtonType",
    EXRECORDLibCtl.ButtonType)
    .Value = "<path>"
    .AddButton "A", 1
    .AddButton "B", 2, RightAlignment
    .ButtonWidth = 20
  End With
  .EndUpdate
End With
```

The ProgressBarType editor  displays a progress bar. The [Value](#) property indicates the percent value being displayed. The options like **exProgressBarBackColor**, **exProgressBarAlignment** or

**exProgressBarMarkTicker** may be used for a ProgressBarType editor.

ProgressBarType

13

The following sample adds a ProgressBarType editor:

```
With Record1
  .BeginUpdate
  With .Add("ProgressBarType",
    EXRECORDLibCtl.ProgressBarType)
    .Value = 34
  End With
  .EndUpdate
End With
```

It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. The PickEditType editor has a standard edit field associated, that useful for searching items. The [DropDownRows](#) property specifies the maximum number of visible rows into the drop=down list. Use [AddItem](#), [InsertItem](#) method to add predefined values to the drop down list. The editor displays the caption of the item that matches [Value](#) value. The item's icon is also displayed if it exists.



The following sample adds a PickEditType editor:

PickEditType

14

```
With Record1
  .BeginUpdate
  With .Add("DropDownType",
    EXRECORDLibCtl.PickEditType)
    .AddItem 0, "Single Bed", 1
    .AddItem 1, "Double Bed", 2
    .AddItem 2, "Apartment", 3
    .AddItem 3, "Suite", 4
    .AddItem 4, "Royal Suite", 5
```



```

.Value = "Apartment"
End With
.EndUpdate
End With

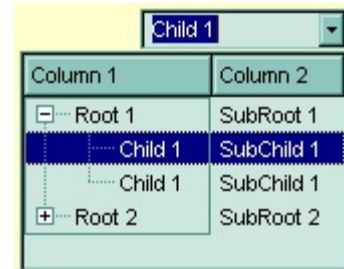
```

LinkEditType

15

The LinkEditType control allows your application to edit and display hyperlink addresses.

The control support ActiveX hosting. An UserEditorType editor can host an ActiveX control. Use the [UserEditor](#) method to create the inner ActiveX control. The



[UserEditorOleEvent](#) event is fired each time when an inner ActiveX control fires an event. Use the [UserEditorObject](#) property to access the inner ActiveX control.

The following sample adds an UserEditorType editor ( the sample adds an inner [Exontrol.ComboBox](#) control ):

UserEditorType

16

```

With Record1
.BeginUpdate
  With .Add("UserEditorType",
EXRECORDLibCtl.UserEditorType)
    .UserEditor "Exontrol.ComboBox", ""
    With .UserEditorObject()
      Dim h As Long
      .BeginUpdate
      .IntegralHeight = True
      .LinesAtRoot = -1
      .ColumnAutoResize = True
      .MinWidthList = 164
      .MinHeightList = 164
      .Columns.Add "Column 1"
      .Columns.Add "Column 2"
      h = .Items.AddItem(Array("Root 1",
"SubRoot 1"))
    End With
  End With
End With

```

```

        .Items.InsertItem h, , Array("Child 1",
"SubChild 1")
        .Items.InsertItem h, , Array("Child 1",
"SubChild 1")
        .Items.ExpandItem(h) = True
        h = .Items.AddItem(Array("Root 2",
"SubRoot 2"))
        .Items.InsertItem h, , Array("Child 1",
"SubChild 1")
        .EndUpdate
    End With
End With
.EndUpdate
End With

```

You can include a color selection control in your applications via the ColorListType editor, also. 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 a new color to your color list editor. You can the **exColorListShowName** to allow the editor displays the color's name.



ColorListType

17

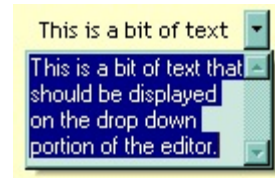
The following sample adds a ColorListType editor:

```

With Record1
    .BeginUpdate
    With .Add("ColorListType",
EXRECORDLibCtl.ColorListType)
        .Option(exColorListShowName) = True
        .Value = RGB(0, 0, 255)
    End With
    .EndUpdate
End With

```

It provides a multiple lines edit control that's displayed into a drop down window. The [AddItem](#), [InsertItem](#) method has no effect, if the EditType is MemoDropDownType.



- The Editor.[Option](#)( **exMemoDropDownWidth** ) specifies the width ( in pixels ) of the MemoDropDownType editor when it is dropped.
- The Editor.[Option](#)( **exMemoDropDownHeight** ) specifies the height ( in pixels ) of the MemoDropDownType editor when it is dropped.
- The Editor.[Option](#)( **exMemoDropDownAcceptReturn** ) specifies whether the user closes the MemoDropDownType editor by pressing the ENTER key. If the Editor.[Option](#)( **exMemoDropDownAcceptReturn** ) is True, the user inserts new lines by pressing the ENTER key. The user can close the editor by pressing the CTRL + ENTER key. If the Editor.[Option](#)( **exMemoDropDownAcceptReturn** ) is False, the user inserts new lines by pressing the CTRL + ENTER key. The user can close the editor by pressing the ENTER key.
- The Editor.[Option](#)( **exMemoHScrollBar** ) adds the horizontal scroll bar to a MemoType or MemoDropDownType editor.
- The Editor.[Option](#)( **exMemoVScrollBar** ) adds the vertical scroll bar to a MemoType or MemoDropDownType editor

MemoDropDownType

18

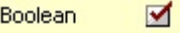
The following sample adds a MemoDropDownType editor:

```
With Record1
    .BeginUpdate
    With .Add("MemoDropDownType",
EXRECORDLibCtl.MemoDropDownType)
```

```

.Option(exMemoVScrollBar) = True
.Value = "This is a bit of text that should be
displayed on the drop down portion of the
editor."
End With
.EndUpdate
End With

```

The CheckValueType editor displays a  Boolean check box based on the [Value](#) property. Use the [CheckImage](#) property to assign a different icons for check box states. You can use the following options:

- **exCheckValue0**, Specifies the check box state being displayed for unchecked state. (long expression, valid values are 0, 1 or 2)
- **exCheckValue1**, Specifies the check box state being displayed for checked state. (long expression, valid values are 0, 1 or 2)
- **exCheckValue2**, Specifies the check box state being displayed for partial checked state. (long expression, valid values are 0, 1 or 2)

CheckValueType

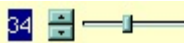
19

The following sample adds a CheckValueType editor:

```

With Record1
.BeginUpdate
With .Add("Boolean",
EXRECORDLibCtl.CheckValueType)
.Option(exCheckValue2) = 1
.Value = True
End With
.EndUpdate
End With

```

Adds a slider control to an editor. Use  the [exSliderWidth](#), [exSliderStep](#), [exSliderMin](#), [exSliderMax](#) options to control the slider

properties. Use the [exSpinStep](#) option to hide the spin control.

The following sample adds a SliderType editor:

SliderType

20

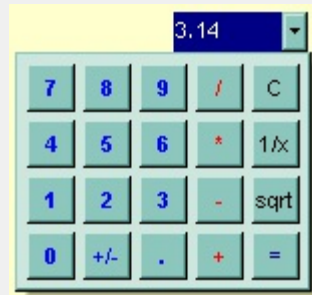
```
With Record1
  .BeginUpdate
  With .Add("Slider",
EXRECORDLibCtl.SliderType)
    .Option(exSpinStep) = 0.1
    .Option(exSliderMax) = 50
    .Value = 34
  End With
  .EndUpdate
End With
```

CalculatorType

21

Adds a drop down calculator to an editor. Use the

[exCalcExecuteKeys](#),  
[exCalcCannotDivideByZero](#),  
[exCalcButtonWidth](#),  
[exCalcButtonHeight](#),  
[exCalcButtons](#),  
[exCalcPictureUp](#), [exCalcPictureDown](#) to specify different options for calculator editor.



```
With Record1
  .BeginUpdate
  With .Add("Calculator",
EXRECORDLibCtl.CalculatorType)
    .Value = 3.14
  End With
  .EndUpdate
End With
```

All editors support the following options:

- **exLeftArrow**, Disables focusing a new cell if the user presses the left arrow key while editing.

- **exRightArrow**, Disables focusing a new cell if the user presses the right arrow key while editing.
- **exUpArrow**, Disable focusing a new cell if the user presses the up arrow key while editing.
- **exDownArrow**, Disable focusing a new cell if the user presses the down arrow key while editing.
- **exHomeKey**, Disable focusing a new cell if the user presses the home key while editing.
- **exEndKey**, Disables focusing a new cell if the user presses the end key while editing.
- **exPageUpKey**, Disable focusing a new cell if the user presses the page up key while editing.
- **exKeepSelBackColor**. Keeps the selection background color while editor is visible

# constants LayoutEnum

The LayoutEnum expression specifies how the fields are arranged in the control's client area. Use the [Layout](#) property to arrange the fields in the control. Use the [LabelSize](#) property to specify the width of the label. Use the [FieldWidth](#), [FieldHeight](#) properties to specify the size of the fields.

Name	Value	Description
exLeftToRight	0	Arranges the fields from the left to the right side of the control.
exTopToBottom	1	Arranges the fields from the top to the bottom side of the control.
exCustomLayout	61440	Customizes the position of fields on the page. The <a href="#">CustomLayout</a> property appends relative position of the fields, when Layout property is exCustomLayout.

# constants InplaceAppearanceEnum

Defines the editor's appearance. Use the [Appearance](#) property to change the editor's appearance. Use the [PopupAppearance](#) property to define the appearance of the editor's drop-down window, if it exists.

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: <b>[+/-]digit</b> , where <b>digit</b> 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: <b>[+/-]digit[.digit[[e/E/d/D][+/-]digit]]</b> , where digit is any combination of <b>digit</b> characters. Use the <a href="#">exEditDecimalSymbol</a> 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 <b>[+/-]digit[.digit]</b> . Use the <a href="#">exEditDecimalSymbol</a> 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



# constants PictureBoxDisplayEnum

Specifies how a picture object is displayed.

Name	Value	Description
UpperLeft	0	Aligns the picture to the upper left corner.
UpperCenter	1	Centers the picture on the upper edge.
UpperRight	2	Aligns the picture to the upper right corner.
MiddleLeft	16	Aligns horizontally the picture on the left side, and centers the picture vertically.
MiddleCenter	17	Puts the picture on the center of the source.
MiddleRight	18	Aligns horizontally the picture on the right side, and centers the picture vertically.
LowerLeft	32	Aligns the picture to the lower left corner.
LowerCenter	33	Centers the picture on the lower edge.
LowerRight	34	Aligns the picture to the lower right corner.
Tile	48	Tiles the picture on the source.
Stretch	49	The picture is resized to fit the source.

# constants ScrollBarsEnum

The ScrollBarsEnum type indicates whether the control displays the horizontal or the vertical scroll bar. The [ScrollBars](#) property specifies whether the control adds scrollbars to the control.

Name	Value	Description
exNoScroll	0	No scroll bars are shown
exHorizontal	1	Only horizontal scroll bars are shown.
exVertical	2	Only vertical scroll bars are shown.
exBoth	3	Both horizontal and vertical scroll bars are shown.

# constants UVisualThemeEnum

The UVisualThemeEnum 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
exSliderVisualTheme	16	exSliderVisualTheme
exSpinVisualTheme	32	exSpinVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exProgressVisualTheme	128	exProgressVisualTheme
exCalculatorVisualTheme	256	exCalculatorVisualTheme

# 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
<a href="#">Add</a>	Adds or replaces a skin object to the control.
<a href="#">Clear</a>	Removes all skins in the control.
<a href="#">Remove</a>	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	<p>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.</p>
Skin as Variant	<p>A string expression that indicates:</p> <ul style="list-style-type: none"><li>• an Windows XP Theme part, it should start with "XP:". For instance the <b>"XP:Header 1 2"</b> 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.</li><li>• 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 <u>"CP:n l t r b"</u>, 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 <b>"CP:1 4 0 -4 0"</b>, 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.</li></ul>

- the path to the skin file ( \*.ebn ). The [Exontrol's exButton](#) component installs a skin builder that should be used to create new skins
- 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 [BeginUpdate](#) and [EndUpdate](#) methods to maintain performance while do multiple changes to the control. Use the [Refresh](#) method to refresh the control.

**The identifier you choose for the skin is very important to be used in the background properties like explained bellow.** Shortly, the color properties uses 4 bytes ( DWORD, double WORD, and so on ) to hold a RGB value. More than that, the first byte ( most significant byte in the color ) is used only to specify system color. if the first bit in the byte is 1, the rest of bits indicates the index of the system color being used. So, we use the last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. So, since the 7 bits can cover 127 values, excluding 0, we have 126 possibilities to store an identifier in that byte. This way, a DWORD expression indicates the background color stored in RRGGBB format and the index of the skin ( ID parameter ) in the last 7 bits in the high significant byte of the color. For instance, the BackColor = BackColor Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the



object that BackColor is applied.

# method Appearance.Clear ()

Removes all skins in the control.

Type	Description
------	-------------

Use the Clear method to clear all skins from the control. Use the [Remove](#) method to remove a specific skin. 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.

# 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

The Editor object holds information about an editor. Use the [Add](#) or [DataSource](#) method to add new editors to the control. The Editor object may display an icon, a custom size picture, an HTML label and window editor.

The Editor object supports the following properties and methods:

Name	Description
<a href="#">AddButton</a>	Adds a new button to the editor with specified key and aligns it to the left or right side of the editor.
<a href="#">AddItem</a>	Adds a new item to the editor's list.
<a href="#">Appearance</a>	Retrieves or sets the editor's appearance.
<a href="#">BackColor</a>	Specifies the editor's background color.
<a href="#">ButtonWidth</a>	Specifies the width of the buttons in the editor.
<a href="#">Caption</a>	Retrieves the caption of the field.
<a href="#">ClearButtons</a>	Clears the buttons collection.
<a href="#">ClearItems</a>	Clears the items collection.
<a href="#">DropDown</a>	Displays the drop down list.
<a href="#">DropDownAlignment</a>	Retrieves or sets a value that indicates the item's alignment in the editor's drop-down list.
<a href="#">DropDownAutoWidth</a>	Retrieves or sets a value that indicates whether the editor's drop-down window list is automatically computed to fit the entire list.
<a href="#">DropDownMinWidth</a>	Specifies the minimum drop-down list width if the DropDownAutoWidth is False.
<a href="#">DropDownRows</a>	Retrieves or sets a value that indicates the maximum number of visible rows in the editor's drop- down list.
<a href="#">DropDownVisible</a>	Retrieves or sets a value that indicates whether the editor's drop down button is visible or hidden.
<a href="#">EditType</a>	Retrieves or sets a value that indicates the type of the editor.
<a href="#">ExpandAll</a>	Expands all items in the editor's list.
<a href="#">ExpandItem</a>	Expands or collapses an item in the editor's list.
<a href="#">FindItem</a>	Finds an item given its value or caption.
<a href="#">ForeColor</a>	Specifies the editor's foreground color.

<a href="#">Image</a>	Retrieves or sets a value that indicates the index of the editor's icon.
<a href="#">Index</a>	Retrieves the index of the editor.
<a href="#">InsertItem</a>	Inserts a child item to the editor's list.
<a href="#">ItemToolTip</a>	Gets or sets the text displayed when the mouse pointer hovers over a predefined item.
<a href="#">Key</a>	Retrieves the editor's key.
<a href="#">Label</a>	Specifies the editor's label.
<a href="#">LabelAlignment</a>	Specifies the alignment of the label relative to the field.
<a href="#">LabelBackColor</a>	Specifies the label's background color.
<a href="#">LabelForeColor</a>	Specifies the label's foreground color.
<a href="#">Locked</a>	Determines whether the editor is locked or unlocked.
<a href="#">Mask</a>	Retrieves or sets a value that indicates the mask used by the editor.
<a href="#">MaskChar</a>	Retrieves or sets a value that indicates the character used for masking.
<a href="#">Numeric</a>	Specifies whether the editor enables numeric values only.
<a href="#">Option</a>	Specifies an option for the editor.
<a href="#">PartialCheck</a>	Retrieves or sets a value that indicates whether the associated check box has two or three states.
<a href="#">Picture</a>	Assigns a custom size picture to an editor.
<a href="#">PopupAppearance</a>	Retrieves or sets a value that indicates the drop-down window's appearance.
<a href="#">Position</a>	Retrieves or sets a value that indicates the editor's position.
<a href="#">RemoveButton</a>	Removes a button given its key.
<a href="#">RemoveItem</a>	Removes an item from the editor's predefined values list.
<a href="#">SortItems</a>	Sorts the list of items in the editor.
<a href="#">ToolTip</a>	Specifies a tooltip being displayed when the cursor hover the editor's label.
<a href="#">UserData</a>	Gets or sets the user-definable data for the current editor.
<a href="#">UserEditor</a>	Specifies the control's identifier and the control's runtime license key when EditType is UserEditor.
<a href="#">UserEditorObject</a>	Gets the user editor object when EditType is UserEditor.

[Value](#)

Retrieves or sets the field's value.

[Visible](#)

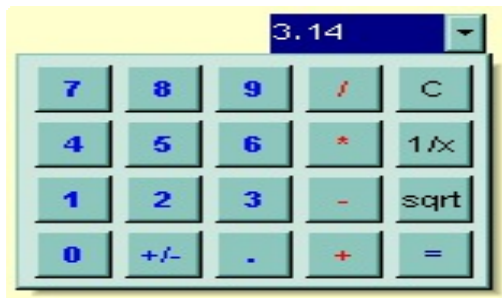
Retrieves or sets a value that indicates whether the editor is visible or hidden.

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

Adds a new button to the editor with specified key and aligns it to the left or right side of the editor.

Type	Description
Key as Variant	A Variant value that indicates the button's key. The <a href="#">ButtonClick</a> event passes this value to Key parameter
Image as Variant	A long expression that indicates the index of button's icon. The index is valid for Images collection. By default the button has no icon associated. Use the <a href="#">Images</a> property to assign a list of icons to the control.
Align as Variant	An <a href="#">AlignmentEnum</a> expression that defines the button's alignment.
ToolTip as Variant	A string expression that indicates the the button's tooltip description. The tooltip shows up when cursor hovers the button. The ToolTip parameter may include built-in HTML tags.
ToolTipTitle as Variant	A string expression that indicates the tooltip's title.
ShortcutKey as Variant	A short expression that indicates the shortcut key being used to simulate clicking the button. The lower byte indicates the code of the virtual key, and the higher byte indicates the states for SHIFT, CTRL and ALT keys ( last insignificant bits in the higher byte ). The ShortcutKey expression could be $256 * ( ( \text{shift} ? 1 : 0 ) + ( \text{ctrl} ? 2 : 0 ) + ( \text{alt} ? 4 : 0 ) ) + \text{vbKeyCode}$ , For instance, a combination like CTRL + F3 is $256 * 2 + \text{vbKeyF3}$ , SHIFT + CTRL + F2 is $256 * (1 + 2) + \text{vbKeyF2}$ , and SHIFT + CTRL + ALT + F5 is $256 * (1 + 2 + 4) + \text{vbKeyF5}$ .

Use the AddButton method to add multiple buttons to the editor. Make sure that you are using unique keys for the buttons in the same editor, else the previous button is replaced. The editor doesn't allow two buttons with the same key. Use the [ButtonWidth](#) property to set the button's width. If the user clicks on one of the editor buttons, the [ButtonClick](#) event is fired. Use the [RemoveButton](#) method to remove a button that was previously added using the AddButton method. Use the [ClearButtons](#) method to clear the entire collection of buttons added with AddButton method. Use the [Images](#) property to assign a list of icons to the control.



The following VB sample adds some buttons to a CalculatorType editor:

```
With Record1
    .BeginUpdate
    .FieldHeight = 20
    .Images
    "gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

    .BackColor = vbWhite
    With .Add("Calculator", CalculatorType)
        .Value = 3.14
        .ButtonWidth = 20
        .AddButton "A", 1, LeftAlignment
        .AddButton "B", 2, LeftAlignment
        .AddButton "C", 1, RightAlignment
        .AddButton "D", 2, RightAlignment
    End With
    .EndUpdate
End With
```

The following VC sample adds some buttons to a CalculatorType editor:

```
ColeVariant vtMissing; vtMissing.vt = VT_ERROR;
m_record.Images( ColeVariant(
    "gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn
));
CEditor editor = m_record.Add(ColeVariant("Calculator"), /*CalculatorType*/ 21, vtMissing
);
editor.SetValue( ColeVariant( 3.14 ) );
editor.SetButtonWidth(18);
editor.AddButton( ColeVariant( "A" ), ColeVariant( (long) 1 ), ColeVariant( (long) 0 ),
vtMissing, vtMissing, vtMissing );
```



```
editor.AddButton( COleVariant( "B" ), COleVariant( (long) 2 ), COleVariant( (long) 1 ),  
vtMissing, vtMissing, vtMissing );  
editor.AddButton( COleVariant( "C" ), COleVariant( (long) 1 ), COleVariant( (long) 1 ),  
vtMissing, vtMissing, vtMissing );  
editor.AddButton( COleVariant( "D" ), COleVariant( (long) 2 ), vtMissing, vtMissing,  
vtMissing, vtMissing );
```

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

Adds a new item to the editor's list.

Type	Description
Value as Long	A long expression that defines a predefined value.
Caption as String	A string expression that indicates the caption for the Value. The Caption supports HTML format.
Image as Variant	A long expression that indicates the index of the item's icon.

Use the AddItem method to add new items to the editor's predefined list. Use the [InsertItem](#) method to insert child items to the editor's predefined list ( **DropDownListType** editor ). If the AddItem method uses a Value already defined, the old item is replaced. The AddItem method has effect for the following type of editors: **DropDownType**, **DropDownListType**, **PickEditType**, and **CheckListType**. Check each [EditType](#) value for what Value argument should contain. Use the [RemoveItem](#) method to remove a particular item from the predefined list. Use the [ClearItems](#) method to clear the entire list of predefined values. Use the [SortItems](#) to sort the items. Use the [ItemToolTip](#) property to assign a tooltip to a predefined item into a drop down list. Call the [Refresh](#) method to update the editor's value, if it depends on a predefined list of items ( drop down editors ). Use the [ItemToolTip](#) property to assign a tooltip to the item.

The Caption property supports the following built-in HTML tags:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" 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</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggbb> ... </fgcolor>** displays text with

a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.

- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&quot;**; ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the

text such as: Text with subscript

- **<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 **<font>** 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 "**<font ;18><gra FFFFFFFF;1;1>gradient-center</gra></font>**" 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 **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><out 000000><fgcolor=FFFFFF>outlined</fgcolor></out></font>**" 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 **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><sha>shadow</sha></font>**" generates the following picture:

shadow

or "**<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>**" gets:

outline anti-aliasing

The following VB sample adds some checks to a CheckListType editor:

```
With Record1
    .BeginUpdate
    With .Add("CheckListType", CheckListType)
```

```
.AddItem &H1, "ReadOnly", 1
.AddItem &H2, "Hidden", 2
.AddItem &H4, "System", 3
.AddItem &H10, "Directory", 4
.AddItem &H20, "Archive", 5
.AddItem &H80, "Normal", 7
.AddItem &H100, "Temporary", 8
.Value = &H1 + &H2
End With
.EndUpdate
End With
```

The following VC sample add some checks to a CheckListType editor:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_record.Add(COleVariant("CheckListType"), /*CheckListType*/ 6,
vtMissing );
editor.AddItem( 0x01, "ReadOnly", vtMissing );
editor.AddItem( 0x02, "Hidden", vtMissing );
editor.AddItem( 0x04, "System", vtMissing );
editor.AddItem( 0x10, "Directory", vtMissing );
editor.AddItem( 0x20, "Archive", vtMissing );
editor.AddItem( 0x80, "Normal", vtMissing );
editor.AddItem( 0x100, "Temporary", vtMissing );
editor.SetValue( COleVariant( (long)(0x01 + 0x02) ) );
m_record.Refresh();
```

# property Editor.Appearance as InplaceAppearanceEnum

Retrieves or sets the editor's appearance.

Type	Description
InplaceAppearanceEnum	An <a href="#">InplaceAppearanceEnum</a> expression that defines the editor's appearance

Use the Appearance property to change the editor's border style. Use the [PopupAppearance](#) property to define the appearance for editor's drop-down window, if it exists. By default, the editor's Appearance is NoApp.

# property Editor.BackgroundColor as Color

Specifies the editor's background color.

Type	Description
Color	A color expression that indicates the background color of the editor.

Use the BackColor property to change the editor's background color. Use the [LabelBackColor](#) property to change the background color of the label of the editor. Use the [ForeColor](#) property to change the editor's foreground color. Use the <bgcolor> HTML tag to specify a background color for parts of the editor's label. Use the [Label](#) property to specify the editor's label. Use the [BackColor](#) property to specify the control's background color.



The following sample assign different background colors for label and the editor as seeing in the screen shot:

```
With Record1
  .BeginUpdate
  With .Add("Label (red)", DropDownType)
    .Value = "Editor (blue)"
    .BackColor = vbBlue
    .LabelBackColor = vbRed
    .ForeColor = vbWhite
    .Position = 0
  End With
  .EndUpdate
End With
```

# property Editor.ButtonWidth as Long

Specifies the width of the buttons in the editor.

Type	Description
Long	A long expression that indicates the width of the buttons in pixels.

Use the ButtonWidth property to change the button's width. By default, the ButtonWidth property is 13 pixels. Use the [AddButton](#) method to add multiple buttons to the editor. Use the [ClearButtons](#) method to clear the editor's buttons collection. If the ButtonWidth property is 0, the editor displays no buttons. The [FieldHeight](#) property specifies the height of the label/editor/field. Use the [DropDownVisible](#) property to hide the editor's drop-down button.



## property Editor.Caption as String

Retrieves the caption of the field.

Type	Description
String	A string expression that specifies the editor's caption.

Use the Caption property to get the editor's caption. Use the [Value](#) property to get the editor's value. The Caption property of the editor may be different than the Value property like follows. For instance, if we have a DropDownListType editor, the Caption property gets the caption of the item being selected, and the Value property gets a long expression that identifies the value of the item. The [Label](#) property gets the editor's label. Use the [FindItem](#) property to find an item based on its value.

The following VB sample prints the label, caption and the value of the editor from the cursor:

```
Private Sub Record1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim e As EXRECORDLibCtl.Editor
    Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not e Is Nothing Then
        Debug.Print "Label: " & e.Label & " Caption: """" & e.Caption & """" Value: " & e.Value
    End If
End Sub
```

The following VC sample prints the label, caption and the value of the editor from the cursor:

```
void OnMouseMoveRecord1(short Button, short Shift, long X, long Y)
{
    CEditor editor = m_record.GetEditorFromPoint( X, Y );
    if ( editor.m_lpDispatch != NULL )
    {
        TCHAR szOutput[1024];
        wsprintf( szOutput, "Label: %s Caption: \"%s\" Value: %s\n",
(LPCTSTR)editor.GetLabel(), (LPCTSTR)editor.GetCaption(), (LPCTSTR)V2S(
&editor.GetValue() ) );
        OutputDebugString( szOutput );
    }
}
```



# method Editor.ClearButtons ()

Clears the buttons collection.

Type	Description
------	-------------

Use the ClearButtons property to clear the editor's collection of buttons. Use the [AddButton](#) method to add multiple buttons to the editor. Use the [ButtonWidth](#) property to specify the width of the buttons. Use the [RemoveButton](#) method to remove a single button. The control fires the [ButtonClick](#) event when user clicks a button.

# method Editor.ClearItems ()

Clears the items collection.

Type	Description
------	-------------

The ClearItems method clears the predefined values added using [AddItem](#), [InsertItem](#) methods. Use the [RemoveItem](#) method to remove a particular item. Use the [DropDownVisible](#) property to hide the drop-down window. Use the [Refresh](#) method to update the editor's content, if it depends on predefined values. Use the [Value](#) property to update the editor's value.

## method Editor.DropDown ()

Displays the drop down list.

Type	Description
------	-------------

The DropDown method shows the drop down portion of the editor. The method has effect for editors like: **DropDownType**, **DropDownListType**, **PickEditType**, and **CheckListType**.

The following VB sample shows the drop down portion of the current editor if the user releases the F2 key:

```
Private Sub Record1_KeyUp(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) Then
        With Record1
            If Not .Focus Is Nothing Then
                .Focus.DropDown
            End If
        End With
    End If
End Sub
```

The following VC sample shows the drop down portion of the current editor if the user releases the F2 key:

```
void OnKeyUpRecord1(short FAR* KeyCode, short Shift)
{
    if ( *KeyCode == VK_F2 )
    {
        CEditor editor = m_record.GetFocus();
        editor.DropDown();
    }
}
```

# property Editor.DropDownAlignment as AlignmentEnum

Retrieves or sets a value that indicates the item's alignment in the editor's drop-down list.

Type	Description
<a href="#">AlignmentEnum</a>	An AlignmentEnum expression that indicates the item's alignment into the editor's drop-down list.

Use the DropDownAlignment property to align the items in the editor's drop-down list. The property has effect only for editors of drop down type. By default, the items in the drop down portion of the editor are left aligned. Use the DropDownAlignment property to right align the items in the drop down portion of the editor.

# property Editor.DropDownAutoWidth as Boolean

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

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

Use the DropDownAutoWidth property to specify when you let the control computes the drop-down list width, or whenever the width is specified by the [DropDownMinWidth](#) property. By default, the DropDownAutoWidth property is True.

# property Editor.DropDownMinWidth as Long

Specifies the minimum drop-down list width if the DropDownAutoWidth is False.

Type	Description
Long	A long expression that specifies the minimum drop- down list width if the DropDownAutoWidth is False

The DropDownMinWidth property has no effect if the [DropDownAutoWidth](#) property is True.



# property Editor.DropDownRows as Long

Retrieves or sets a value that indicates the maximum number of visible rows in the editor's drop- down list.

Type	Description
Long	A long expression that indicates the maximum number of visible rows in the editor's drop- down list.

Use the DropDownRows property to specify the maximum number of visible rows in the editor's drop-down list. By default, the DropDownRows property is set to 7. The DropDownRows property has effect for the following types: DropDownType, DropDownListType, PickEditType, CheckListType and FontType.

# property Editor.DropDownVisible as Boolean

Retrieves or sets a value that indicates whether the editor's drop down button is visible or hidden.

Type	Description
Boolean	A boolean value that indicates whether the editor's drop down button is visible or hidden.

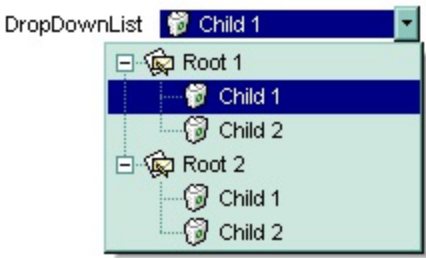
By default, the DropDownVisible property is True. Use the DropDownVisible property to hide the editor's drop-down button. Use the [ButtonWidth](#) property to hide the editor buttons.

# property Editor.EditType as EditTypeEnum

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

Type	Description
<a href="#">EditTypeEnum</a>	An EditTypeEnum expression that specifies the type of the editor.

Use the EditType property to set the editor's type. Use the [Add](#) method to specify the editor's type when adding the editor to the control. The EditType property is set when using the [DataSource](#) property according to the record set field's type. For instance, if we have a record set field of date type, the EditType property is set to DateType. Use the [UserEditor](#) method to specify the program identifier when EditType property is UserEditorType. Use the [UserEditorObject](#) property to access the inner ActiveX control when EditType property is UserEditorType. Use the [Value](#) property to specify the editor's value. Use the [Option](#) property to define options for a specific type of editor. Use the [AddItem](#) and [InsertItem](#) methods to add new items to the drop down portion of the editor. Use the [AddButton](#) method to add new buttons to the editor.



The following VB sample adds an editor of drop down list type:

```
With Record1
    .BeginUpdate
    .BackColor = vbWhite
    .LabelSize = 80
    .Images
    "gBJJgBAICAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

    With .Add("DropDownList", EXRECORDLibCtl.DropDownListItemType)
        .DropDownAutoWidth = False
        .AddItem 0, "Root 1", 1
        .InsertItem 1, "Child 1", 2, 0
        .InsertItem 2, "Child 2", 2, 0
```

```

.AddItem 3, "Root 2", 1
.InsertItem 4, "Child 1", 2, 3
.InsertItem 5, "Child 2", 2, 3
.ExpandAll
End With
.EndUpdate
End With

```

The following VC sample adds an editor of drop down list type:

```

CVariant vtMissing; vtMissing.vt = VT_ERROR;
m_record.Images(CVariant("gBJgBAICAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaI

CEditor editor = m_record.Add(CVariant("DropDownList"), /*DropDownListType*/ 3,
vtMissing );
editor.SetDropDownAutoWidth( FALSE );
editor.AddItem( 0, "Root 1", COleVariant((long)1) );
editor.InsertItem(1, "Child 1", COleVariant((long)2), COleVariant( long(0) ) );
editor.InsertItem(2, "Child 2", COleVariant((long)2), COleVariant( long(0) ) );
editor.AddItem( 3, "Root 2", COleVariant((long)1) );
editor.InsertItem(4, "Child 1", COleVariant((long)2), COleVariant( long(3) ) );
editor.InsertItem(5, "Child 2", COleVariant((long)2), COleVariant( long(3) ) );
editor.ExpandAll();
m_record.Refresh();

```

The following VC sample adds an [Exontrol.ComboBox](#) ActiveX control:

```

#import "c:\winnt\system32\ExComboBox.dll"
#import "c:\winnt\system32\ExRecord.dll"

BOOL sInstalled(BSTR strProgID)
{
    CLSID clsid = CLSID_NULL;
    HRESULT hResult = E_POINTER;
    if (SUCCEEDED( hResult = CLSIDFromProgID( strProgID, &clsid; ) ))
    {
        IDispatch* pObject = NULL;
        if ( SUCCEEDED( hResult = CoCreateInstance( clsid, NULL, CLSCTX_ALL, IID_IDispatch,

```

```
reinterpret_cast(&pObject;) ) ) )
```

```
{  
    pObject->Release();  
    return TRUE;  
}  
}  
return FALSE;  
}
```

```
#define v(x) _variant_t( x )
```

```
CString strObject( "Exontrol.ComboBox" );  
COleVariant vtMissing; vtMissing.vt = VT_ERROR;  
m_record.BeginUpdate();  
m_record.SetLabelSize( 110 );  
CEditor editor = m_record.Add( COleVariant( "ActiveX" ),  
EXRECORDLib::UserEditorType, vtMissing );  
editor.SetPosition( 2 );  
if ( !IsInstalled( strObject.AllocSysString() ) )  
{  
    CString strFormat;  
    strFormat.Format( "\"%s\" is not installed.", (LPCSTR)strObject );  
    editor.SetValue( COleVariant( strFormat ) );  
    editor.SetForeColor( RGB( 255, 0, 0 ) );  
}  
else  
{  
    // Creates the exComboBox control. https://www.exontrol.com/excombobox.jsp  
    editor.UserEditor( strObject, "" );  
    if ( EXCOMBOBOXLib::IComboBoxPtr spComboBox = editor.GetUserEditorObject() )  
    {  
        spComboBox->BeginUpdate();  
        spComboBox->BackColorEdit = GetSysColor( COLOR_MENU );  
        spComboBox->IntegralHeight = true;  
        spComboBox->ColumnAutoResize = true;  
        spComboBox->LinesAtRoot = EXCOMBOBOXLib::exLinesAtRoot;  
        spComboBox->MinHeightList = 164;
```

```
spComboBox->MinWidthList = 264;
spComboBox->MarkSearchColumn = false;
spComboBox->DrawGridLines = EXCOMBOBOXLib::exAllLines;
spComboBox->FilterBarDropDownHeight = -150;
spComboBox->Alignment = EXCOMBOBOXLib::RightAlignment;
EXCOMBOBOXLib::lColumnsPtr spColumns = spComboBox->Columns;
spColumns->Add("Column 1");
spColumns->Add("Column 2");
EXCOMBOBOXLib::lColumnPtr spColumn = spColumns->Add("Column 3");
spColumn->DisplayFilterButton = true;
EXCOMBOBOXLib::lItemsPtr spltems = spComboBox->Items;
long h = spltems->AddItem( v( "Root 1" ) );
spltems->CellCaption[v(h)][v((long)1)] = v("SubChild 1");
spltems->CellCaption[v(h)][v((long)2)] = v("SubChild 2");
long h1 = spltems->InsertItem( h, vtMissing, v( "Child 1" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 1.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 1.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
h1 = spltems->InsertItem( h, vtMissing, v( "Child 2" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 2.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 2.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
spltems->put_ExpandItem( h, TRUE );
```

```
h = spltems->AddItem( v( "Root 2" ) );
spltems->CellCaption[v(h)][v((long)1)] = v("SubChild 1");
spltems->CellCaption[v(h)][v((long)2)] = v("SubChild 2");
h1 = spltems->InsertItem( h, vtMissing, v( "Child 1" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 1.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 1.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
h1 = spltems->InsertItem( h, vtMissing, v( "Child 2" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 2.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 2.2");
```

```

spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
spltems->put_ExpandItem( h, TRUE );

spComboBox->Value = "Root 1";
spComboBox->EndUpdate();

}
}
m_record.EndUpdate();

```

The following VB sample adds a Forms.ComboBox ActiveX control:

```

With Record1
    .BeginUpdate
    With .Add("ActiveX", EXRECORDLibCtl.UserEditorType)
        .UserEditor "Forms.ComboBox.1", ""
        With .UserEditorObject()
            .AddItem "Item 1"
            .AddItem "Item 2"
        End With
    End With
    .EndUpdate
End With

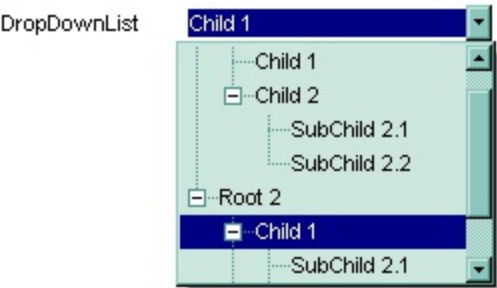
```

# method Editor.ExpandAll ()

Expands all items in the editor's list.

Type	Description
------	-------------

The ExapndAll method expands all items in the editor's drop down list. Use the [InsertItem](#) method to insert child items to the editor's drop down list. Use the [ExpandItem](#) method to expand programmatically an item. The ExpandAll method has effect only if the [EditType](#) property is **DropDownListType**. Use the [DropDownAutoWidth](#) property to let the control computes the width of the drop down portion so all items hit the drop down client's area.



The following VB sample expands all items in the DropDownListType editor:

```
With Record1
  .BeginUpdate
  With .Add("DropDownList", EXRECORDLibCtl.DropDownListItemType)
    .DropDownAutoWidth = False
    .AddItem 0, "Root 1"
    .InsertItem 1, "Child 1", , 0
    .InsertItem 2, "Child 2", , 0
    .InsertItem 3, "SubChild 2.1", , 2
    .InsertItem 4, "SubChild 2.2", , 2
    .AddItem 5, "Root 2"
    .InsertItem 6, "Child 1", , 5
    .InsertItem 7, "Child 2", , 5
    .InsertItem 8, "SubChild 2.1", , 6
    .InsertItem 9, "SubChild 2.2", , 6
    .ExpandAll
    .Value = 6
  End With
  .EndUpdate
End With
```



The following VC sample expands all items in aDropDownListType editor:

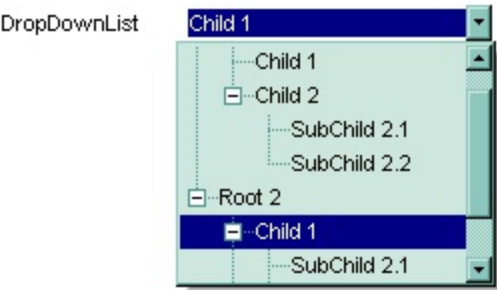
```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_record.Add(COleVariant("DropDownList"), /*DropDownListType*/ 3,
vtMissing );
editor.SetDropDownAutoWidth( FALSE );
editor.AddItem( 0, "Root 1", vtMissing );
editor.InsertItem( 1, "Child 1", vtMissing, COleVariant( long(0) ) );
editor.InsertItem( 2, "Child 2", vtMissing, COleVariant( long(0) ) );
editor.InsertItem( 3, "SubChild 2.1", vtMissing, COleVariant( long(2) ) );
editor.InsertItem( 4, "SubChild 2.2", vtMissing, COleVariant( long(2) ) );
editor.AddItem( 5, "Root 2", vtMissing );
editor.InsertItem( 6, "Child 1", vtMissing, COleVariant( long(0) ) );
editor.InsertItem( 7, "Child 2", vtMissing, COleVariant( long(0) ) );
editor.InsertItem( 8, "SubChild 1.1", vtMissing, COleVariant( long(6) ) );
editor.InsertItem( 9, "SubChild 1.2", vtMissing, COleVariant( long(6) ) );
editor.SetValue( COleVariant( (long)6 ) );
editor.ExpandAll();
```

# property Editor.ExpandItem(Value as Variant) as Boolean

Expandes or collapses an item in the editor's list.

Type	Description
Value as Variant	A long expression that indicates the value of the item being expanded, a string expression that indicates the caption of the item being expanded.
Boolean	A boolean expression that indicates whether the item is expanded or collapsed.

By default, the items are collapsed. Use the ExpandItem to expand a specified item. Use the [ExpandAll](#) method to expand all items in the editor. Use the [InsertItem](#) method to insert a child item to your drop down editor. The ExpandItem property has effect only if the [EditType](#) property is DropDownListType.



The following VB sample expands all items in the DropDownListType editor:

```
With Record1
  .BeginUpdate
  With .Add("DropDownList", EXRECORDLibCtl.DropDownListType)
    .DropDownAutoWidth = False
    .AddItem 0, "Root 1"
    .InsertItem 1, "Child 1", , 0
    .InsertItem 2, "Child 2", , 0
    .InsertItem 3, "SubChild 2.1", , 2
    .InsertItem 4, "SubChild 2.2", , 2
    .AddItem 5, "Root 2"
    .InsertItem 6, "Child 1", , 5
    .InsertItem 7, "Child 2", , 5
    .InsertItem 8, "SubChild 2.1", , 6
    .InsertItem 9, "SubChild 2.2", , 6
    .ExpandAll
```

```
.Value = 6  
End With  
.EndUpdate  
End With
```

The following VC sample expands all items in a DropDownListType editor:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;  
CEditor editor = m_record.Add(COleVariant("DropDownList"), /*DropDownListType*/ 3,  
vtMissing );  
editor.SetDropDownAutoWidth( FALSE );  
editor.AddItem( 0, "Root 1", vtMissing );  
editor.InsertItem( 1, "Child 1", vtMissing, COleVariant( long(0) ) );  
editor.InsertItem( 2, "Child 2", vtMissing, COleVariant( long(0) ) );  
editor.InsertItem( 3, "SubChild 2.1", vtMissing, COleVariant( long(2) ) );  
editor.InsertItem( 4, "SubChild 2.2", vtMissing, COleVariant( long(2) ) );  
editor.AddItem( 5, "Root 2", vtMissing );  
editor.InsertItem( 6, "Child 1", vtMissing, COleVariant( long(0) ) );  
editor.InsertItem( 7, "Child 2", vtMissing, COleVariant( long(0) ) );  
editor.InsertItem( 8, "SubChild 1.1", vtMissing, COleVariant( long(6) ) );  
editor.InsertItem( 9, "SubChild 1.2", vtMissing, COleVariant( long(6) ) );  
editor.SetValue( COleVariant( (long)6 ) );  
editor.ExpandAll();
```

# 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 string expression that indicates the caption of the item, if the Value is a long expression, a long expression that indicates the item's value if Value is a string expression.

Use the FindItem property look for an item in the drop down list editor. The FindItem property retrieves an empty ( VT\_EMPTY ) value if no item is found. Use the [AddItem](#) or [InsertItem](#) method to add new items to the drop down list editor.

The following VB sample prints the caption of the item with the value 6:

```
With Record1
    .BeginUpdate
    With .Add("DropDownList", EXRECORDLibCtl.DropDownListType)
        .DropDownAutoWidth = False
        .AddItem 0, "Root 1"
        .InsertItem 1, "Child 1", , 0
        .InsertItem 2, "Child 2", , 0
        .InsertItem 3, "SubChild 2.1", , 2
        .InsertItem 4, "SubChild 2.2", , 2
        .AddItem 5, "Root 2"
        .InsertItem 6, "Child 1", , 5
        .InsertItem 7, "Child 2", , 5
        .InsertItem 8, "SubChild 2.1", , 6
        .InsertItem 9, "SubChild 2.2", , 6
        Debug.Print .FindItem(6)
    End With
    .EndUpdate
End With
```

The sample displays the "Child 1" string that's the caption for the item with the value 6.

The following VB sample displays the value of the item with the caption "SubChild 2.1":

With Record1

.BeginUpdate

With .Add("DropDownList", EXRECORDLibCtl.DropDownListBoxType)

.DropDownAutoWidth = False

.AddItem 0, "Root 1"

.InsertItem 1, "Child 1", , 0

.InsertItem 2, "Child 2", , 0

**.InsertItem 3, "SubChild 2.1", , 2**

.InsertItem 4, "SubChild 2.2", , 2

.AddItem 5, "Root 2"

.InsertItem 6, "Child 1", , 5

.InsertItem 7, "Child 2", , 5

.InsertItem 8, "SubChild 2.1", , 6

.InsertItem 9, "SubChild 2.2", , 6

**Debug.Print .FindItem("SubChild 2.1")**

End With

.EndUpdate

End With

The sample displays 3 as being the value of the "SubChild 2.1" item.

The following VC sample looks for the item with the value 6:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}
```

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
```

```

CEditor editor = m_record.Add(COLEVariant("DropDownList"), /*DropDownListType*/ 3,
vtMissing );
editor.SetDropDownAutoWidth( FALSE );
editor.AddItem( 0, "Root 1", vtMissing );
editor.InsertItem( 1, "Child 1", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 2, "Child 2", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 3, "SubChild 2.1", vtMissing, COLEVariant( long(2) ) );
editor.InsertItem( 4, "SubChild 2.2", vtMissing, COLEVariant( long(2) ) );
editor.AddItem( 5, "Root 2", vtMissing );
editor.InsertItem( 6, "Child 1", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 7, "Child 2", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 8, "SubChild 1.1", vtMissing, COLEVariant( long(6) ) );
editor.InsertItem( 9, "SubChild 1.2", vtMissing, COLEVariant( long(6) ) );

COLEVariant vtItem = editor.GetFindItem( COLEVariant( (long)6 ) );
OutputDebugString( V2S( &vtItem ) );

```

The sample displays in the output window the "Child 1" string that indicates the caption of the item with the value 6.

The following VC sample looks for an item given its caption:

```

COLEVariant vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_record.Add(COLEVariant("DropDownList"), /*DropDownListType*/ 3,
vtMissing );
editor.SetDropDownAutoWidth( FALSE );
editor.AddItem( 0, "Root 1", vtMissing );
editor.InsertItem( 1, "Child 1", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 2, "Child 2", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 3, "SubChild 2.1", vtMissing, COLEVariant( long(2) ) );
editor.InsertItem( 4, "SubChild 2.2", vtMissing, COLEVariant( long(2) ) );
editor.AddItem( 5, "Root 2", vtMissing );
editor.InsertItem( 6, "Child 1", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 7, "Child 2", vtMissing, COLEVariant( long(0) ) );
editor.InsertItem( 8, "SubChild 1.1", vtMissing, COLEVariant( long(6) ) );
editor.InsertItem( 9, "SubChild 1.2", vtMissing, COLEVariant( long(6) ) );

COLEVariant vtItem = editor.GetFindItem( COLEVariant( "SubChild 2.2" ) );

```

```
OutputDebugString( V2S( &vtItem ) );
```

The sample displays in the output window the 4 value that corresponds to the "SubChild 2.2" item.

# property Editor.ForeColor as Color

Specifies the editor's foreground color.

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

Use the ForeColor property to specify the editor's foreground color. Use the [LabelForeColor](#) property to specify the label's foreground color. Use the [ForeColor](#) property to specify the foreground color for the entire control. Use the <fgcolor> HTML tag to specify a foreground color for parts of the editor's label. Use the [Label](#) property to specify the editor's label.



The following sample assign different background colors for label and the editor as seeing in the screen shot:

```
With Record1
  .BeginUpdate
  With .Add("Label (red)", DropDownType)
    .Value = "Editor (blue)"
    .BackColor = vbBlue
    .LabelBackColor = vbRed
    .ForeColor = vbWhite
    .Position = 0
  End With
  .EndUpdate
End With
```



# property Editor.Image as Long

Retrieves or sets a value that indicates the index of the editor's icon.

Type	Description
Long	A long expression that indicates the index of the icon being displayed in the editor's label. The control's list of icons is 1 based.

Use the Image property to assign a 16x16 icon to the editor. Use the [Images](#) method to assign a list of icons to the control. By default, the Image property is 0, no icon is displayed. Use the [Picture](#) property to assign a custom size picture to the editor. Use the [Value](#) property to assign a value to the editor. Use the [Label](#) property to specify the label of the editor. Use the <img> tag to insert icons or custom size pictures inside the field's label.



The following VB sample loads a list of icons from a BASE64 encoded string:

```
Dim s As String
s =
"gD/bD/cAACBKlxJlwAAIBAEMiAAf7bABDAADAQCjEaAcdAkeAoIAoFAgEAoKA4HAWlBgKB
s = s +
"jjmVZuG4JBrHiTJuj4Z4YHoNICDiNoUFWsBnHIOouFeTw8HWexLHwWJxD6LYrHgTxEnETovG
s = s +
"jB6jRBeNkc4VAHDWBsLEFw6R3B2E8DIAIChSDwAyIEEA4wKjiB8DRZwghDD4GWOIOolxDC
s = s +
"BAIlgagHgEBvEwGQAACQQBXFcH4ZwPwJD8DmOMKguwPgRFYPQJolwti6DCHUNYoB0iM
s = s +
"IUYqx3iWCEKcOwdhABGBwBwBgAAVjqDmNIVI6RfjqDKs4VwjQogMGGBcJoRhOAVBkgSBq
s = s +
"hPh9gbBth2gHASBPgwBvB1hoBNBbADACghv6gAAPA4FnBcAfhKgngvAkRIhsiAg= ="
With Record1
    .BeginUpdate
```

```

.BackColor = vbWhite
.LabelSize = 166
.Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

With .Add("Label", EXRECORDLibCtl.ReadOnly)
    .Image = 1
    .Picture = s
    .Value = "Editor"
End With
.EndUpdate
End With

```

The following VC sample loads a list of icons from a BASE64 encoded string:

```

CString s(
"gd/bD/cAACBKlxlwAAIBAEMiAAf7bABDAADAQCjEaAcdAkeAoIAoFAgEAoKA4HAWlBgKB
");
s = s +
"jjmVZuG4JBrHiTJuj4Z4YHoNICDiNoUFWsBnHlOouFeTw8HWexLHwWJxD6LYrHgTxEnETovG

s = s +
"jB6jRBeNkc4VAHDWBsLEFw6R3B2E8DIAIChSDwAyIEEA4wKjiB8DRZwghDD4GWOIOolxDC

s = s +
"BAIlgagHgEBvEwGQAACQQBXFcH4ZwPwJD8DmOMKguwPgRFYPQJolwti6DCHUNYoB0iM

s = s +
"lUYqx3iWCEKcOwdhABGBwBwBgAAVjqDmNIVI6RfjqDKs4VwjQogMGGBcJoRhOAVBkgSBq

s = s +
"hPh9gbBth2gHASBPgwBvB1hoBNBbADACghv6gAAPA4FnBcAfhKgngvAkRIhsiAg = ";

m_record.BeginUpdate();
m_record.Images( COleVariant(
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn
));

```

```
m_record.SetBackColor( RGB(255,255,255) );  
m_record.SetLabelSize(166);  
COleVariant vtMissing; vtMissing.vt = VT_ERROR;  
CEditor editor = m_record.Add(COleVariant("Label"), /*ReadOnly*/ 0, vtMissing );  
editor.SetImage( 1 );  
editor.SetValue( COleVariant( "Editor" ) );  
editor.SetPicture( COleVariant( (LPCTSTR)s ) );  
m_record.EndUpdate();
```

# property Editor.Index as Long

Retrieves the index of the editor.

Type	Description
Long	A long expression that specifies the index of the editor.

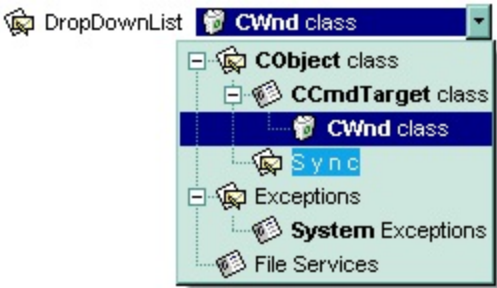
Use the Index property to retrieve the index of the editor in the control's collection of [Editor](#) objects. Use the [Key](#) property to identify an editor. Use the [Position](#) property to specify the editor's position. Use the [Visible](#) property to hide an editor. By default, the first editor added has the Index property on 0. The Index property of the editor is updated as soon as an editor is removed. Use the [Item](#) property to access an editor by index or by key. Use the [ItemByPosition](#) property to access an editor giving its position.

# method Editor.InsertItem (Value as Long, Caption as String, [Image as Variant], [Parent as Variant])

Inserts a child item to the editor's list.

Type	Description
Value as Long	A long expression that indicates the value of the item
Caption as String	A string expression that indicates the caption of the item. The Caption parameter may include built-in HTML tags like explained bellow.
Image as Variant	A long expression that indicates the index of the icon being displayed.
Parent as Variant	A long expression that defines the value of the parent item.

Use the InsertItem to insert child items to the editor's predefined list. Use the [AddItem](#) method to add new items to the editor's drop down list. Use the [ExpandItem](#) property to expand an item. Use the [ExpandAll](#) items to expand all items. Use the [ItemTooltip](#) property to assign a tooltip to a predefined item into a drop down editor. Use the [RemoveItem](#) method to remove an item.



The following VB sample inserts few items to a drop down list editor:

```
With Record1
    .BeginUpdate
    .BackColor = vbWhite
    .Images
    ("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUIk8pIUrlktl0vmExi
    .Images
    ("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUIk8pIUrlktl0vmExi
    .Images
```

("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlk8plUrlktl0vmExn

```
With .Add("DropDownList", EXRECORDLibCtl.DropDownListItemType)
```

```
.Image = 1
```

```
.DropDownAutoWidth = False
```

```
.AddItem 1, "CObject class", 1
```

```
.InsertItem 2, "CCmdTarget class", 2, 1
```

```
.InsertItem 3, "CWnd class", 3, 2
```

```
.InsertItem 6, "S y n c", 1, 1
```

```
.AddItem 4, "Exceptions", 1
```

```
.InsertItem 7, "System Exceptions", 2, 4
```

```
.AddItem 5, "File Services", 2
```

```
.ExpandAll
```

```
.Value = 3
```

```
End With
```

```
.EndUpdate
```

```
End With
```

The following VC sample inserts some items to a drop down list editor:

```
ColeVariant vtMissing; vtMissing.vt = VT_ERROR;
```

```
m_record.BeginUpdate();
```

```
m_record.Images( COleVariant(
```

```
"gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlk8plUrlktl0vmExn  
));
```

```
m_record.Images( COleVariant(
```

```
"gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlk8plUrlktl0vmExn  
));
```

```
m_record.Images( COleVariant(
```

```
"gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlk8plUrlktl0vmExn  
));
```

```
m_record.SetBackColor( RGB(255,255,255) );
```

```
CEditor editor = m_record.Add(COleVariant("DropDownList"), /*DropDownListItemType*/ 3,  
vtMissing );
```

```
editor.SetDropDownAutoWidth( FALSE );
```

```
editor.AddItem(1, "CObject class", COleVariant( (long)1 ) );
```

```
editor.InsertItem( 2, "CCmdTarget class", COleVariant( (long)2 ), COleVariant( (long)1 ) );
```

```

editor.InsertItem( 3, "CWnd class", COleVariant( (long)3 ), COleVariant( (long)2 ) );
editor.InsertItem( 6, "S y n c", COleVariant( (long)1 ), COleVariant( (long)1 ) );
editor.AddItem(4, "Exceptions", COleVariant( (long)1 ) );
editor.InsertItem(7, "System Exceptions", COleVariant( (long)2 ), COleVariant( (long)4 ) );
editor.AddItem(5, "File Services", COleVariant( (long)2 ) );
editor.ExpandAll();
editor.SetValue( COleVariant( (long)3 ) );
m_record.EndUpdate();

```

The Caption parameter may include the following built-in HTML tags:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" 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</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rr gg bb> ... </fgcolor>** or **<fgcolor=rr gg bb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rr gg bb> ... </bgcolor>** or **<bgcolor=rr gg bb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rr gg bb> ... </solidline>** or **<solidline=rr gg bb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rr gg bb> ... </dotline>** or **<dotline=rr gg bb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires

<solidline> or <dotline>).

- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;** ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<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 **<font>** 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 "**<font ;18><gra FFFFFFFF;1;1>**gradient-center**</gra></font>**" 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 `<font>` HTML tag can be used to define the height of the font. For instance the "`<font ;31><out 000000><fgcolor=FFFFFF>outlined</fgcolor></out></font>`" 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 `<font>` HTML tag can be used to define the height of the font. For instance the "`<font ;31><sha>shadow</sha></font>`" generates the following picture:

shadow

or "`<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>`" gets:

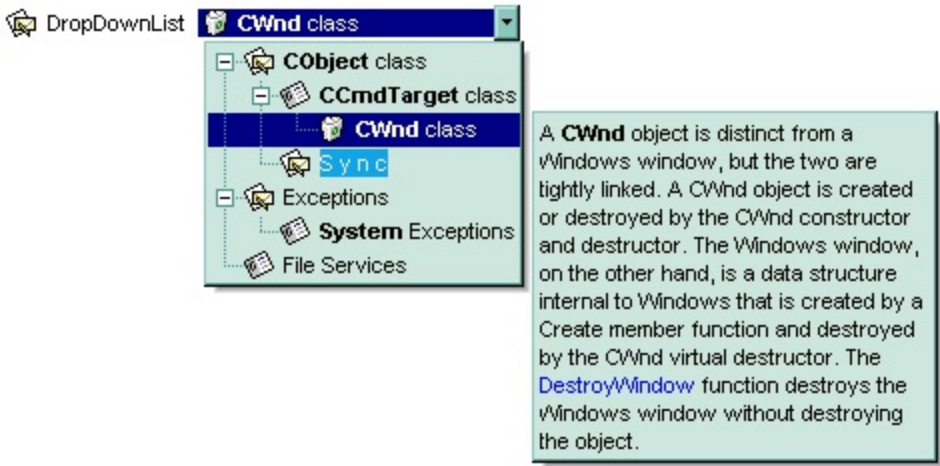
outline anti-aliasing

# property Editor.ItemToolTip(Value as Variant) as String

Gets or sets the text displayed when the mouse pointer hovers over a predefined item.

Type	Description
Value as Variant	A long expression that indicates the value of the item whose tooltip is accessed, a string expression that indicates the caption of the item whose tooltip is accessed.
String	A string expression that may include HTML tags, that indicates the text being displayed when the mouse hovers the item.

Use the ItemToolTip property to assign a tooltip for a drop down list value. Use the [AddItem](#) or [InsertItem](#) methods to insert new items to the drop down predefined list. Use the [ToolTip](#) property to assign a tooltip to an editor.



The following VB sample adds a tooltip for a predefined value:

```
With Record1
    .BeginUpdate
    .BackColor = vbWhite
    .Images
    ("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUl k8p lUr lkt l0vmExi

    .Images
    ("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUl k8p lUr lkt l0vmExi

    .Images
    ("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUl k8p lUr lkt l0vmExi
```

```
With .Add("DropDownList", EXRECORDLibCtl.DropDownListBoxType)
```

```
.Image = 1
```

```
.DropDownAutoWidth = False
```

```
.AddItem 1, "CObject class", 1
```

```
.InsertItem 2, "CCmdTarget class", 2, 1
```

```
.InsertItem 3, "CWnd class", 3, 2
```

.ItemToolTip(3) = "A **CWnd** object is distinct from a Windows window, but the two are tightly linked. A CWnd object is created or destroyed by the CWnd constructor and destructor. The Windows window, on the other hand, is a data structure internal to Windows that is created by a Create member function and destroyed by the CWnd virtual destructor. The DestroyWindow function destroys the Windows window without destroying the object. "

```
.InsertItem 6, "S y n c", 1, 1
```

```
.AddItem 4, "Exceptions", 1
```

```
.InsertItem 7, "System Exceptions", 2, 4
```

```
.AddItem 5, "File Services", 2
```

```
.ExpandAll
```

```
.Value = 3
```

```
End With
```

```
.EndUpdate
```

```
End With
```

The following VC sample adds a tooltip to a predefined value:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
```

```
m_record.BeginUpdate();
```

```
m_record.Images( COleVariant(
```

```
"gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlK8pIUrlktl0vmExn  
));
```

```
m_record.Images( COleVariant(
```

```
"gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlK8pIUrlktl0vmExn  
));
```

```
m_record.Images( COleVariant(
```

```
"gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjlBAEijUlK8pIUrlktl0vmExn  
));
```

```
m_record.SetBackColor( RGB(255,255,255) );
```

```

CEditor editor = m_record.Add(COleVariant("DropDownList"), /*DropDownListType*/ 3,
vtMissing );
editor.SetDropDownAutoWidth( FALSE );
editor.AddItem(1, "Object class", COleVariant( (long)1 ) );
editor.InsertItem( 2, "CCmdTarget class", COleVariant( (long)2 ), COleVariant( (long)1 ) );
editor.InsertItem( 3, "CWnd class", COleVariant( (long)3 ), COleVariant( (long)2 ) );
editor.SetItemToolTip( COleVariant( (long)3 ), "A CWnd object is distinct from a Windows
window, but the two are tightly linked. A CWnd object is created or destroyed by the
CWnd constructor and destructor. The Windows window, on the other hand, is a data
structure internal to Windows that is created by a Create member function and destroyed
by the CWnd virtual destructor. The DestroyWindow function destroys the Windows
window without destroying the object. " );
editor.InsertItem( 6, "S y n c", COleVariant( (long)1 ), COleVariant( (long)1 ) );
editor.AddItem(4, "Exceptions", COleVariant( (long)1 ) );
editor.InsertItem(7, "System Exceptions", COleVariant( (long)2 ), COleVariant( (long)4 ) );
editor.AddItem(5, "File Services", COleVariant( (long)2 ) );
editor.ExpandAll();
editor.SetValue( COleVariant( (long)3 ) );
m_record.EndUpdate();

```

The ItemToolTip property may include the following built-in HTML tags:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" 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</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.

- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&quot;**; ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<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 <font> 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 "<font ;18><gra FFFFFFFF;1;1>gradient-center</gra></font>" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><out 000000><fgcolor=FFFFFF>outlined</fgcolor></out></font>" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

or "<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>" gets:

outline anti-aliasing

# property Editor.Key as String

Retrieves the editor's key.

Type	Description
String	A string expression that specifies the editor's key.

The Key property specifies the editor's key. The Key property is read only. The Key parameter of the [Add](#) method indicates the key of the editor being added. Use the [Item](#) property to access an editor by key or by its index. The [Index](#) property retrieves the index of the editor. Use the [Position](#) property to specify the position of the editor.

# property Editor.Label as String

Specifies the editor's label.

Type	Description
String	A string expression that indicates the label of the editor.

The Label property specifies the editor's label. Use the Label property to change the editor's label. Use the [LabelSize](#) property to specify the width of the label. Use the [Value](#) property to specify the value of the editor. Use the [Caption](#) property to retrieve the caption of the editor. Use the [Add](#) method to specify the editor's label at adding time. Use the [Image](#) property to assign an icon to an editor. Use the <img> tag to insert icons inside the field's label. Use the [Picture](#) property to assign a picture to an editor. Use the [LabelForeColor](#) property to specify the label's foreground color. Use the [LabelBackColor](#) property to specify the label's background color. Use the [LabelAlignment](#) property to align the label.

The Label property may include built-in HTML tags like follows:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" 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</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggbb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggbb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggbb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The



rr/gg/bb represents the red/green/blue values of the color in hexa values.

- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggbb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<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 **<font>** 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 "**<font ;18><gra**

FFFFFF;1;1>gradient-center</gra></font>" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><out 000000>

<fgcolor=FFFFFF>outlined</fgcolor></out></font>" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

or "<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>" gets:

outline anti-aliasing

# property Editor.LabelAlignment as AlignmentEnum

Specifies the alignment of the label relative to the field.

Type	Description
<a href="#">AlignmentEnum</a>	An AlignmentEnum expression that indicates the label's alignment.

Use the LabelAlignment property to align the label. By default, the label is aligned to the left.

# property Editor.LabelBackColor as Color

Specifies the label's background color.

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

Use the LabelBackColor property to change the background color of the label of the editor. Use the [BackColor](#) property to change the editor's background color. Use the [ForeColor](#) property to change the editor's foreground color. Use the <bgcolor> HTML tag to specify a background color for parts of the editor's label. Use the [Label](#) property to specify the editor's label. Use the [BackColor](#) property to specify the control's background color.



The following sample assign different background colors for label and the editor as seeing in the screen shot:

```
With Record1
  .BeginUpdate
  With .Add("Label (red)", DropDownType)
    .Value = "Editor (blue)"
    .BackColor = vbBlue
    .LabelBackColor = vbRed
    .ForeColor = vbWhite
    .Position = 0
  End With
  .EndUpdate
End With
```

# property Editor.LabelForeColor as Color

Specifies the label's foreground color.

Type	Description
Color	A color expression that specifies the editor's label foreground color.

Use the LabelForeColor property to specify the label's foreground color. Use the [ForeColor](#) property to specify the editor's foreground color. Use the [ForeColor](#) property to specify the foreground color for the entire control. Use the <fgcolor> HTML tag to specify a foreground color for parts of the editor's label. Use the [Label](#) property to specify the editor's label. Use the [LabelBackColor](#) property to specify the background color of the editor's label.



The following sample assign different background colors for label and the editor as seeing in the screen shot:

```
With Record1
  .BeginUpdate
  With .Add("Label (red)", DropDownType)
    .Value = "Editor (blue)"
    .BackColor = vbBlue
    .LabelBackColor = vbRed
    .ForeColor = vbWhite
    .Position = 0
  End With
  .EndUpdate
End With
```

# property Editor.Locked as Boolean

Determines whether the editor is locked or unlocked.

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

Use the Locked property to lock an editor. By default, the Locked property is False. If the Locked property is True, the editor is locked. For instance, if the [EditType](#) property is EditType, and Locked property is True, the edit control is read-only, and so the user can type new text inside. Use the [Visible](#) property to hide the editor. Use the [Enabled](#) property to disable the control.

# 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 defines the editor's mask.

Use the Mask property to filter characters during data input. Use the Mask property to control the entry of many types of formatted information such as telephone numbers, social security numbers, IP addresses, license keys etc. The Mask property has effect for the following edit types: DropDownType, SpinType, DateType, MaskType, FontType, PickEditType. Call the [Refresh](#) method to update the editor's mask.

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:

Rule	Name	Description
#	Digit	Masks a digit character. [0-9]
x	Hexa Lower	Masks a lower hexa character. [0-9],[a-f]
X	Hexa Upper	Masks a upper 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 letter. [a-z]
>	Alphabetic Upper	Masks an upper letter. [A-Z]
*	Any	Mask 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 by brackets []. For instance, the [abcA-C] mask any character: a,b,c,A,B,C

The following VB sample adds an IP editor:

With Record1

.BeginUpdate

With .Add("IP", EXRECORDLibCtl.MaskType)

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

.Value = "193.226.40.161"

End With

.EndUpdate

End With

The following VC sample adds a Phone editor:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
```

```
CEditor editor = m_record.Add(COleVariant("Phone"), /*MaskType*/ 8, vtMissing );
```

```
editor.SetMask("(###) ### - ####");
```

```
editor.SetValue( COleVariant( "(0744) 845 - 2835" ) );
```

```
m_record.Refresh();
```



# 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.

Use the MaskChar property to change the default masking character, which is '\_'. The MaskChar property has effect only if the Mask property is not empty, and the mask is applicable to the editor's type. Use the [Mask](#) property to specify the editor's mask.

# property Editor.Numeric as NumericEnum

Specifies whether the editor enables numeric values only.

Type	Description
<a href="#">NumericEnum</a>	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 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 <a href="#">EditorOptionEnum</a>	An EditorOptionEnum expression that indicates the editor's option being changed.
Variant	A Variant expression that indicates the value for editor's option

Use the Option property to change the options for an editor.

The following VB sample adds a password editor:

```
With Record1
    .BeginUpdate
    With .Add("Password", EXRECORDLibCtl.EditType)
        .Option(exEditPassword) = True
        .Value = "pass"
    End With
    .EndUpdate
End With
```

The following VC sample adds a password editor:

```
ColeVariant vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_record.Add(ColeVariant("Password"), /*EditType*/ 1, vtMissing );
editor.SetOption( /*exEditPassword*/ 18, ColeVariant( (long)TRUE ) );
editor.SetValue( ColeVariant( "pass" ) );
```

# 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.

# property Editor.Picture as Variant

Assigns a custom size picture to an editor.

Type	Description
Variant	A Picture object that indicates the picture being assigned, a string expression that may indicate the path to a picture file or a string expression that indicates the base64 encoded string that holds a picture object. Use the <a href="#">eximages</a> tool to save your picture as base64 encoded format.

The Picture property assigns a custom size picture to an editor. Use the [Image](#) property to assign an icon to the editor. The picture is displayed on the editor's label, if it's width is less than the label's width. Use the [LabelSize](#) property to enlarge the label's size. Use the [Picture](#) property to display a picture to the control's background. You can load a transparent picture ( a GIF file with transparency but set ) in order to display a transparent picture.



The following VB sample loads a list of icons from a BASE64 encoded string:

```
Dim s As String
s =
"gd/bD/cAACBKlxJlwAAIBAEMiAAf7bABDAADAQCjEaAcdAkeAoIAoFAgEAoKA4HAWlBgKB
s = s +
"jjmVZuG4JBrHiTJuj4Z4YHoNICDiNoUFWsBnHIOouFeTw8HWexLHwWJxD6LYrHgTxEnETovG
s = s +
"jB6jRBeNkc4VAHDWBsLEFw6R3B2E8DIAIChSDwAyIEEA4wKjiB8DRZwghDD4GWOIOolxDC
s = s +
"BAIlgagHgEBvEwGQAACQQBXFcH4ZwPwJD8DmOMKguwPgRFYPQJolwti6DCHUNYoB0iM
s = s +
"IUYqx3iWCEKcOwdhABGBwBwBgAAVjqDmNIVI6RfjqDKs4VwjQogMGGBcJoRhOAVBkgSBq
s = s +
"hPh9gbBth2gHASBPgwBvB1hoBNBbADACghv6gAAPA4FnBcAfhKgngvAkRIhsiAg = ="
```

With Record1

.BeginUpdate

.BackColor = vbWhite

.LabelSize = 166

.Images

"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm

With .Add("**Label**", EXRECORDLibCtl.ReadOnly)

.Image = 1

.Picture = s

.Value = "Editor"

End With

.EndUpdate

End With

The following VC sample loads a list of icons from a BASE64 encoded string:

CString s(

"gD/bD/cAACBKlxJlwAAIBAEMiAAf7bABDAADAQCjEaAcdAkeAoIAoFAgEAoKA4HAWlBgKB  
);

s = s +

"jjmVZuG4JBrHiTJuj4Z4YHoNICDiNoUFWsBnHIOouFeTw8HWexLHwWJxD6LYrHgTxEnETovG

s = s +

"jB6jRBeNkc4VAHDWBsLEFw6R3B2E8DIAIChSDwAyIEEA4wKjiB8DRZwghDD4GWOIOolxDC

s = s +

"BAIlgagHgEBvEwGQAACQQBXFcH4ZwPwJD8DmOMKguwPgRFYPQJolwti6DCHUNYoB0iM

s = s +

"IUyqx3iWCEKcOwdhABGBwBwBgAAVjqDmNIVI6RfjqDKs4VwjQogMGGBcJoRhOAVBkgSBq

s = s +

"hPh9gbBth2gHASBPgwBvB1hoBNBbADACghv6gAAPA4FnBcAfhKgngvAkRIhsiAg==";

m\_record.BeginUpdate();

```
m_record.Images( COleVariant(
"gBJJgBAICAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn
"));
m_record.SetBackColor( RGB(255,255,255) );
m_record.SetLabelSize(166);
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_record.Add(COleVariant("Label"), /*ReadOnly*/ 0, vtMissing );
editor.SetImage( 1 );
editor.SetValue( COleVariant( "Editor" ) );
editor.SetPicture( COleVariant( (LPCTSTR)s ) );
m_record.EndUpdate();
```

# property Editor.PopupAppearance as InplaceAppearanceEnum

Retrieves or sets a value that indicates the drop-down window's appearance.

Type	Description
InplaceAppearanceEnum	An <a href="#">InplaceAppearanceEnum</a> expression that defines the drop-down window's border style.

Use the PopupAppearance property to change the drop-down window's border style. Use the [Appearance](#) property to define the editor's appearance.



# property Editor.Position as Long

Retrieves or sets a value that indicates the editor's position.

Type	Description
Long	A long expression that indicates the position of the editor.

Use the Position property to change the editor's position. Use the [ItemByPosition](#) property to access an Editor by its position. Use the [Visible](#) property to hide an editor. Use the [Key](#) property to identify an editor.

# 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 RemoveButton method to remove a single button. Use the [AddButton](#) method to add multiple buttons to the editor. Use the [ButtonWidth](#) property to specify the width of the buttons. The control fires the [ButtonClick](#) event when user clicks a button. Use the [ClearButtons](#) property to clear the editor's collection of buttons.

# method Editor.RemoveItem (Value as Long)

Removes an item from the editor's predefined values list.

Type	Description
Value as Long	A long expression that indicates the index of the item being removed, or a string expression that indicates the caption of the item being removed.

Use the RemoveItem method to remove an item from the editor's predefined values list. Use the [ClearItems](#) method to clear the entire list of editor items. Use the [DropDownVisible](#) property to hide the editor's drop-down window. Use the [Remove](#) method to remove an editor.

# 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 the items in a drop down editor. Use the [AddItem](#) or [InsertItem](#) method to add new items to the control. Use the [RemoveItem](#) method to remove an item. Use the [ClearItems](#) method to clear the items. Use the [ExpandAll](#) method to expand all items.

# Property Editor.ToolTip as String

Specifies a tooltip being displayed when the cursor hover the editor's label.

Type	Description
String	A string expression that specifies the editor's tooltip.

The ToolTip property specifies the editor's tooltip. The editor's tooltip shows up when the cursor hovers the editor. By default, the ToolTip property is "...". If the ToolTip property is "...", the tooltip appears only if the [Label](#) property is not fully visible.

The ToolTip property may include built-in HTML tags like follows:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" 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</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggbb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggbb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggbb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggbb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires

<solidline> or <dotline>).

- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<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 **<font>** 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 "**<font ;18><gra FFFFFFFF;1;1>**gradient-center**</gra></font>**" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><out 000000><fgcolor=FFFFFF>outlined</fgcolor></out></font>" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

or "<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>" gets:

outline anti-aliasing

# property Editor.UserData as Variant

Gets or sets the user-definable data for the current editor.

Type	Description
Variant	A Variant expression that specifies the editor's extra data.

Use the UserData property to associate an extra data to an editor. Use the [Value](#) property to change the editor's value. Use the [Label](#) property to specify the editor's label. Use the [Caption](#) property to get the editor's caption.



## method Editor.UserEditor (ControlID as String, License as String)

Specifies the control's identifier and the control's runtime license key when EditType is UserEditor.

Type	Description
ControlID as String	A string expression that indicates the control's program identifier. For instance, if you want to use a multiple column combobox as an user editor, the control's identifier could be: "Exontrol.ComboBox".
License as String	Optional. A string expression that indicates the runtime license key in case is it required. It depends on what control are you using.

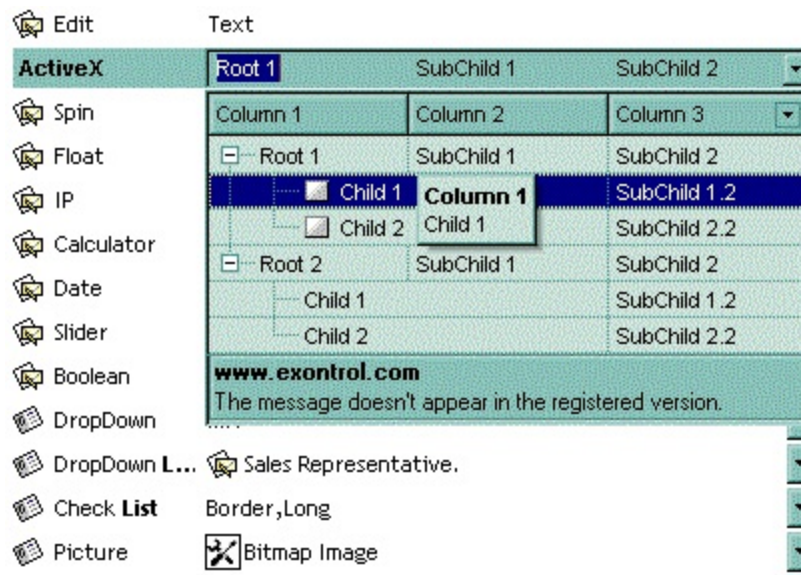
The UserEditor property creates an editor that hosts an inner ActiveX control, based on the ControlID parameter. The UserEditor property has effect only if the [EditType](#) property is UserEditorType. Use the [UserEditorObject](#) property to access the newly created object. The UserEditorObject property is nothing if the control wasn't able to create the user editor based on the ControlID. Also, if the user control requires a runtime license key, and the License parameter is empty or doesn't match, the UserEditorObject property is nothing. The control fires the [UserEditorOleEvent](#) event each time when an user editor fires an event.

The control supports ActiveX hosting, so you can insert any ActiveX component. The ControlID must be formatted in one of the following ways:

- A ProgID such as "Exontrol.ComboBox"
- A CLSID such as "{8E27C92B-1264-101C-8A2F-040224009C02}"
- A reference to an Active document such as "c:\temp\myfile.doc", or "c:\temp\picture.gif"
- A fragment of HTML such as "MSHTML:<HTML><BODY>This is a line of text</BODY></HTML>"
- A fragment of XML

**The look and feel of the inner ActiveX control depends on the identifier you are using, and the version of the library that implements the ActiveX control, so you need to consult the documentation of the inner ActiveX control you are inserting inside the exRecord control.** Unfortunately, You need to contact the vendor for particular ActiveX controls, because we can't provide documentation for any ActiveX control you might use. We can provide documentation only for our components.

Use the [UserEditor](#) method to create an inner ActiveX control.



The following VB sample adds an [Exontrol.ComboBox](http://www.exontrol.com) control and displays the events being fired by inner ActiveX control:

### Option Explicit

```
Private Function isInstalled(ByVal s As String) As Boolean
```

```
On Error GoTo Error
```

```
    CreateObject (s)
```

```
    isInstalled = True
```

```
    Exit Function
```

```
Error:
```

```
    isInstalled = False
```

```
End Function
```

```
Private Sub Form_Load()
```

```
    With Record1
```

```
        .BeginUpdate
```

```
        With .Add("ActiveX", UserEditorType)
```

```
            .Position = 2
```

```
            Dim progID As String
```

```
            progID = "Exontrol.ComboBox"
```

```
            If Not (isInstalled(progID)) Then
```

```
                .Value = "" & progID & "" is not installed."
```

```
                .ToolTip = .Value
```

```
                .ForeColor = vbRed
```

```
            Else
```

.UserEditor progID, ""

.LabelBackColor = SystemColorConstants.vbMenuBar

' Accesses the inside ActiveX control, in our case an ExComboBox control.

<https://www.exontrol.com/excombobox.jsp>

With .UserEditorObject()

.BeginUpdate

.BackColorEdit = SystemColorConstants.vbMenuBar

.IntegralHeight = True

.ColumnAutoResize = True

.LinesAtRoot = True

.MinHeightList = 164

.MinWidthList = 264

.MarkSearchColumn = False

.FilterBarDropDownHeight = -150

.DrawGridLines = True

.Alignment = 0

With .Columns

.Add "Column 1"

.Add "Column 2"

With .Add("Column 3")

.DisplayFilterButton = True

End With

End With

With .Items

Dim h, h1

h = .AddItem(Array("Root 1", "SubChild 1", "SubChild 2"))

h1 = .InsertItem(h, , Array("Child 1", "SubChild 1.1", "SubChild 1.2"))

.CellMerge(h1, 0) = 1

.CellHasCheckBox(h1, 0) = True

h1 = .InsertItem(h, , Array("Child 2", "SubChild 2.1", "SubChild 2.2"))

.CellMerge(h1, 0) = 1

.CellHasCheckBox(h1, 0) = True

.ExpandItem(h) = True

h = .AddItem(Array("Root 2", "SubChild 1", "SubChild 2"))

h1 = .InsertItem(h, , Array("Child 1", "SubChild 1.1", "SubChild 1.2"))

.CellMerge(h1, 0) = 1

h1 = .InsertItem(h, , Array("Child 2", "SubChild 2.1", "SubChild 2.2"))

```

        .CellMerge(h1, 0) = 1
        .ExpandItem(h) = True
    End With
    .Value = "Root 1"
    .EndUpdate
End With
End If
End With
.EndUpdate
End With
End Sub

```

```

Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
On Error Resume Next
    Debug.Print "Event name: " & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print vbTab & "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub

```

The following VC sample adds an [Exontrol.ComboBox](#) control and displays the events being fired by inner ActiveX control:

```

#import "c:\winnt\system32\ExComboBox.dll"
#import "c:\winnt\system32\ExRecord.dll"

CString strObject( "Exontrol.ComboBox" );
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_record.BeginUpdate();
m_record.SetLabelSize( 110 );
CEditor editor = m_record.Add( COleVariant( "ActiveX" ), EXRECORDLib::UserEditorType,

```

```

vtMissing );
editor.SetPosition( 2 );
if ( !isInstalled( strObject.AllocSysString() ) )
{
    CString strFormat;
    strFormat.Format( "\"%s\" is not installed.", (LPCSTR)strObject );
    editor.SetValue( COleVariant( strFormat ) );
    editor.SetForeColor( RGB( 255, 0, 0 ) );
}
else
{
    // Creates the exComboBox control. https://www.exontrol.com/excombobox.jsp
    editor.UserEditor( strObject, "" );
    if ( EXCOMBOBOXLib::IComboBoxPtr spComboBox = editor.GetUserEditorObject() )
    {
        spComboBox->BeginUpdate();
        spComboBox->BackColorEdit = GetSysColor( COLOR_MENU );
        spComboBox->IntegralHeight = true;
        spComboBox->ColumnAutoResize = true;
        spComboBox->LinesAtRoot = EXCOMBOBOXLib::exLinesAtRoot;
        spComboBox->MinHeightList = 164;
        spComboBox->MinWidthList = 264;
        spComboBox->MarkSearchColumn = false;
        spComboBox->DrawGridLines = EXCOMBOBOXLib::exAllLines;
        spComboBox->FilterBarDropDownHeight = -150;
        spComboBox->Alignment = EXCOMBOBOXLib::RightAlignment;
        EXCOMBOBOXLib::IColumnsPtr spColumns = spComboBox->Columns;
        spColumns->Add("Column 1");
        spColumns->Add("Column 2");
        EXCOMBOBOXLib::IColumnPtr spColumn = spColumns->Add("Column 3");
        spColumn->DisplayFilterButton = true;
        EXCOMBOBOXLib::IItemsPtr spltems = spComboBox->Items;
        long h = spltems->AddItem( v( "Root 1" ) );
        spltems->CellCaption[v(h)][v((long)1)] = v("SubChild 1");
        spltems->CellCaption[v(h)][v((long)2)] = v("SubChild 2");
        long h1 = spltems->InsertItem( h, vtMissing, v( "Child 1" ) );
        spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 1.1");
    }
}

```

```

spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 1.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
h1 = spltems->InsertItem( h, vtMissing, v( "Child 2" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 2.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 2.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
spltems->put_ExpandItem( h, TRUE );

```

```

h = spltems->AddItem( v( "Root 2" ) );
spltems->CellCaption[v(h)][v((long)1)] = v("SubChild 1");
spltems->CellCaption[v(h)][v((long)2)] = v("SubChild 2");
h1 = spltems->InsertItem( h, vtMissing, v( "Child 1" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 1.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 1.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
h1 = spltems->InsertItem( h, vtMissing, v( "Child 2" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 2.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 2.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
spltems->put_ExpandItem( h, TRUE );

```

```

spComboBox->Value = "Root 1";
spComboBox->EndUpdate();

```

```

}
}
m_record.EndUpdate();

```

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )

```

```

        return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)
{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
    {
        strOutput += "\tThe event has no arguments.";
    }
    else
    {
        strOutput += "\tThe event has no arguments.\r\n";
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            strOutput += spEvent->GetParam( v(i) )->Name;
            strOutput += " = ";
            strOutput += V2S( &spEvent->GetParam( v(i) )->Value );
            strOutput += "\r\n";
        }
    }
    OutputDebugString( strOutput );
}

```

In C++, the **#import "path-to-ExRecord.dll"** adds a new EXRECORDLib namespace that includes definition for [OleEvent](#) and [OleEventParam](#) classes.

# property Editor.UserEditorObject as Object

Gets the user editor object when EditType is UserEditor.

Type	Description
Object	An ActiveX object being used as an user editor.

Use the UserEditorOpen property to access to the inner ActiveX user editor. Use the [UserEditor](#) property to initialize the inner ActiveX user editor. The UserEditorObject property retrieves the ActiveX control created by the UserEditor method. The type of object returned by the UserEditorObject depends on the ControlID parameter of the UserEditor method. For instance, the type of the created object when UserEditor("Exontrol.ComboBox") is called, is EXCOMBOBOXLibCtl.ComboBox. The UserEditorObject property gets nothing if the UserEditor method fails to create the inner ActiveX control. The control fires the [UserEditorOleEvent](#) event each time when an user editor fires an event.



# property Editor.Value as Variant

Retrieves or sets the field's value.

Type	Description
Variant	A Variant expression that specifies the editor's value.

Use the Value property to get the editor's value. Use the [Caption](#) property to get the editor's caption. The Caption property of the editor may be different than the Value property like follows. For instance, if we have a DropDownListType editor, the Caption property gets the caption of the item being selected, and the Value property gets a long expression that identifies the value of the item. The [Label](#) property gets the editor's label. Use the [FindItem](#) property to find an item based on its value. The Change event is fired when the user alters the editor's content. Use the [AddItem](#) property to add predefined items to a drop down editor (**DropDownType**, **DropDownListType**, **PickEditType**, and **CheckListType**). Use the [EditType](#) property to change the type of the editor. Call the [Refresh](#) method to update the editor's value, if it depends on a predefined list of items ( drop down editors ). The [Change](#) event is called when the uses changes the value for an editor. If the control is bounded to a recordset ( [DataSource](#) property ), the value of the field is automatically updated in the recordset too.

The following VB sample prints the label, caption and the value of the editor from the cursor:

```
Private Sub Record1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim e As EXRECORDLibCtl.Editor
    Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not e Is Nothing Then
        Debug.Print "Label: " & e.Label & " Caption: """" & e.Caption & """" Value: " & e.Value
    End If
End Sub
```

The following VC sample prints the label, caption and the value of the editor from the cursor:

```
void OnMouseMoveRecord1(short Button, short Shift, long X, long Y)
{
    CEditor editor = m_record.GetEditorFromPoint( X, Y );
    if ( editor.m_lpDispatch != NULL )
    {
```

```
TCHAR szOutput[1024];  
wsprintf( szOutput, "Label: %s Caption: \"%s\" Value: %s\n",  
(LPCTSTR)editor.GetLabel(), (LPCTSTR)editor.GetCaption(), (LPCTSTR)V2S(  
&editor.GetValue() ) );  
OutputDebugString( szOutput );  
}  
}
```

# property Editor.Visible as Boolean

Retrieves or sets a value that indicates whether the editor is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the editor is visible or hidden.

Use the Visible property to hide an editor. Use the [Position](#) property to change the order of the editors.

The following VB sample hides the "Handler" editor:

```
Record1("Handler").Visible = False
```

The following VC sample hides the "Handler" editor:

```
m_record.GetItem(COleVariant("Handler")).SetVisible( FALSE )
```

# OleEvent object

The OleEvent object holds information about an event fired by an inner ActiveX control hosted by an [UserEditorType](#) editor. The [UserEditorOleEvent](#) event is fired when an inner ActiveX control fires an event.

Name	Description
<a href="#">CountParam</a>	Retrieves the count of the OLE event's arguments.
<a href="#">ID</a>	Retrieves a long expression that specifies the identifier of the event.
<a href="#">Name</a>	Retrieves the original name of the fired event.
<a href="#">Param</a>	Retrieves an OleEventParam object given either the index of the parameter, or its name.
<a href="#">ToString</a>	Retrieves information about the event.

## property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

Type	Description
Long	A long value that indicates the count of the arguments.

The following VB sample shows how to enumerate the arguments of an OLE event:

```
Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
On Error Resume Next
    Debug.Print "Event name: " & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print vbTab & "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample enumerates the arguments of an OLE event:

```
void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)
{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
    {
        strOutput += "\tThe event has no arguments.";
    }
    else
    {
```

```

strOutput += "\tThe event has no arguments.\r\n";
for ( long i = 0; i < spEvent->CountParam; i++ )
{
    strOutput += spEvent->GetParam( v(i) )->Name;
    strOutput += " = ";
    strOutput += V2S( &spEvent->GetParam( v(i) )->Value);
    strOutput += "\r\n";
}

}
m_output = strOutput;
UpdateData( FALSE );
}

```

The #import "c:\winnt\system32\ExRecord.dll" generates the EXRECORDLib namespace that includes definitions for OleEvent and OleEventParam objects.

# property OleEvent.ID as Long

Retrieves a long expression that specifies the identifier of the event.

Type	Description
Long	A Long expression that defines the identifier of the OLE event.

The identifier of the event could be used to identify a specified OLE event. Use the [Name](#) property of the OLE Event to get the name of the OLE Event. Use the [ToString](#) property to display information about an OLE event. The ToString property displays the identifier of the event after the name of the event in two [] brackets. For instance, the ToString property gets the "KeyDown[-602](KeyCode/Short\* = 9,Shift/Short = 0)" when TAB key is pressed, so the identifier of the KeyDown event being fired by the inside User editor is -602.

## property OleEvent.Name as String

Retrieves the original name of the fired event.

Type	Description
String	A string expression that indicates the event's name.

The Name property indicates the name of the event. Use the [ID](#) property to specify a specified even by its identifier. Use the [ToString](#) property to display information about an OLE event. The ToString property displays the identifier of the event after the name of the event in two [] brackets. For instance, the ToString property gets the "KeyDown[-602] (KeyCode/Short\* = 9,Shift/Short = 0)" when TAB key is pressed, so the identifier of the KeyDown event being fired by the inside User editor is -602. The following VB sample shows how to enumerate the arguments of an OLE event:

```
Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
On Error Resume Next
    Debug.Print "Event name: " & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print vbTab & "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample enumerates the arguments of an OLE event:

```
void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)
{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
```



```

{
    strOutput += "\tThe event has no arguments.";
}
else
{
    strOutput += "\tThe event has no arguments.\r\n";
    for ( long i = 0; i < spEvent->CountParam; i++ )
    {
        strOutput += spEvent->GetParam( v(i) )->Name;
        strOutput += " = ";
        strOutput += V2S( &spEvent->GetParam( v(i) )->Value);
        strOutput += "\r\n";
    }

}
m_output = strOutput;
UpdateData( FALSE );
}

```

The `#import "c:\winnt\system32\ExRecord.dll"` generates the EXRECORDLib namespace that includes definitions for OleEvent and OleEventParam objects.

---

## property OleEvent.Param (item as Variant) as OleEventParam

Retrieves an OleEventParam object given either the index of the parameter, or its name.

Type	Description
item as Variant	A long expression that indicates the argument's index or a string expression that indicates the argument's name.
OleEventParam	An <a href="#">OleEventParam</a> object that contains the name and the value for the argument.

The following VB sample shows how to enumerate the arguments of an OLE event:

```
Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
On Error Resume Next
    Debug.Print "Event name: " & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print vbTab & "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample enumerates the arguments of an OLE event:

```
void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)
{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
    {
```

```

        strOutput += "\tThe event has no arguments.";
    }
    else
    {
        strOutput += "\tThe event has no arguments.\r\n";
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            strOutput += spEvent->GetParam( v(i) )->Name;
            strOutput += " = ";
            strOutput += V2S( &spEvent->GetParam( v(i) )->Value);
            strOutput += "\r\n";
        }

    }
    m_output = strOutput;
    UpdateData( FALSE );
}

```

The `#import "c:\winnt\system32\ExRecord.dll"` generates the EXRECORDLib namespace that includes definitions for OleEvent and OleEventParam objects.

# property OleEvent.ToString as String

Retrieves information about the event.

Type	Description
String	A String expression that shows information about an OLE event. The ToString property gets the information as follows: Name[ID] (Param/Type = Value, Param/Type = Value, ... ). For instance, "KeyDown[-602] (KeyCode/Short* = 9,Shift/Short = 0)" indicates that the KeyDown event is fired, with the identifier -602 with two parameters KeyCode as a reference to a short type with the value 8, and Shift parameter as Short type with the value 0.

Use the ToString property to display information about fired event such us name, parameters, types and values. Using the ToString property you can quickly identifies the event that you should handle in your application. Use the [ID](#) property to specify a specified even by its identifier. Use the [Name](#) property to get the name of the event. Use the [Param](#) property to access a specified parameter using its index or its name.

Displaying ToString property during the OLE Event event may show data like follows:

```
MouseMove[-606](Button/Short = 0,Shift/Short = 0,X/Long = 46,Y/Long = 15)
MouseDown[-605](Button/Short = 1,Shift/Short = 0,X/Long = 46,Y/Long = 15)
KeyDown[-602](KeyCode/Short* = 83,Shift/Short = 0)
KeyPress[-603](KeyAscii/Short* = 115)
Change[2]()
KeyUp[-604](KeyCode/Short* = 83,Shift/Short = 0)
MouseUp[-607](Button/Short = 1,Shift/Short = 0,X/Long = 46,Y/Long = 15)
MouseMove[-606](Button/Short = 0,Shift/Short = 0,X/Long = 46,Y/Long = 15)
```

# OleEventParam object

The OleEventParam holds the name and the value for an event's argument. The [UserEditorOleEvent](#) event is fired when an inner ActiveX control fires an event.

Name	Description
<a href="#">Name</a>	Retrieves the name of the event's parameter.
<a href="#">Value</a>	Retrieves the value of the event's parameter.

## property OleEventParam.Name as String

Retrieves the name of the event's parameter.

Type	Description
String	A string expression that indicates the name of the event's parameter.

The following VB sample shows how to enumerate the arguments of an OLE event:

```
Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
On Error Resume Next
    Debug.Print "Event name: " & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print vbTab & "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample enumerates the arguments of an OLE event:

```
void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)
{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
    {
        strOutput += "\tThe event has no arguments.";
    }
    else
    {
```

```
strOutput += "\tThe event has no arguments.\r\n";
for ( long i = 0; i < spEvent->CountParam; i++ )
{
    strOutput += spEvent->GetParam( v(i) )->Name;
    strOutput += " = ";
    strOutput += V2S( &spEvent->GetParam( v(i) )->Value);
    strOutput += "\r\n";
}

}
m_output = strOutput;
UpdateData( FALSE );
}
```

The #import "c:\winnt\system32\ExRecord.dll" generates the EXRECORDLib namespace that includes definitions for OleEvent and OleEventParam objects.

---

## property OleEventParam.Value as Variant

Specifies the value of the event's parameter.

Type	Description
Variant	A variant value that indicates the value of the event's parameter.

The following VB sample shows how to enumerate the arguments of an OLE event:

```
Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
On Error Resume Next
    Debug.Print "Event name: " & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print vbTab & "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample enumerates the arguments of an OLE event:

```
void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)
{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
    {
        strOutput += "\tThe event has no arguments.";
    }
    else
    {
```



```
strOutput += "\tThe event has no arguments.\r\n";
for ( long i = 0; i < spEvent->CountParam; i++ )
{
    strOutput += spEvent->GetParam( v(i) )->Name;
    strOutput += " = ";
    strOutput += V2S( &spEvent->GetParam( v(i) )->Value);
    strOutput += "\r\n";
}

}
m_output = strOutput;
UpdateData( FALSE );
}
```

The #import "c:\winnt\system32\ExRecord.dll" generates the EXRECORDLib namespace that includes definitions for OleEvent and OleEventParam objects.

---

# Record 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: {656D66AF-1E46-45E3-B1B5-FFE9FB353AC7}. The object's program identifier is: "Exontrol.Record". The /COM object module is: "ExRecord.dll"

Exontrol's new exRecord control is a container component that displays a set of editors added manually or bounded to a table in a database. The exRecord name comes from the record, that's a set of fields that contain related information, in database type systems. The exRecord significantly reduces development time of data components. The Record object supports the following properties and methods:

Name	Description
<a href="#">Add</a>	Adds an editor and returns a reference to the newly created object.
<a href="#">Appearance</a>	Retrieves or sets the control's appearance.
<a href="#">AttachTemplate</a>	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
<a href="#">BackColor</a>	Specifies the control's background color.
<a href="#">Background</a>	Returns or sets a value that indicates the background color for parts in the control.
<a href="#">BeginUpdate</a>	Maintains performance when editors are added to the control one at a time. This method prevents the control from painting until the EndUpdate method is called.
<a href="#">BorderHeight</a>	Sets or retrieves a value that indicates the border height of the control.
<a href="#">BorderWidth</a>	Sets or retrieves a value that indicates the border width of the control.
<a href="#">CheckImage</a>	Retrieves or sets a value that indicates the index of the image for the checkbox fields.
<a href="#">Count</a>	Returns the number of editors in a control.
<a href="#">CustomLayout</a>	Specifies an array of relative positions that are used when the control arranges the fields on the page.
<a href="#">DataSource</a>	Retrieves or sets a value that indicates the data source for object.
<a href="#">EditorFromPoint</a>	Retrieves the editor from point.
<a href="#">Enabled</a>	Enables or disables the control.
<a href="#">EndUpdate</a>	Resumes painting the control after painting is suspended by the BeginUpdate method.

<a href="#">EnsureVisible</a>	Ensures that a field fits the control's client area.
<a href="#">ExecuteTemplate</a>	Executes a template and returns the result.
<a href="#">FieldHeight</a>	Retrieves or sets a value that indicates the height of the field.
<a href="#">FieldWidth</a>	Retrieves or sets a value that indicates the width of the field.
<a href="#">Focus</a>	Specifies the editor that gets the focus.
<a href="#">Font</a>	Retrieves or sets the control's font.
<a href="#">ForeColor</a>	Specifies the control's foreground color.
<a href="#">HBorderField</a>	Returns or sets a value that indicates the distance between two fields on the horizontal axis.
<a href="#">HFieldCount</a>	Sets or gets a value that indicates the number of fields on the horizontal axis.
<a href="#">HTMLPicture</a>	Adds or replaces a picture in HTML captions.
<a href="#">hWnd</a>	Retrieves the control's window handle.
<a href="#">Images</a>	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
<a href="#">ImageSize</a>	Retrieves or sets the size of icons the control displays..
<a href="#">item</a>	Returns an editor based on its index.
<a href="#">ItemByPosition</a>	Returns an editor based on its position.
<a href="#">LabelAlignment</a>	Specifies the alignment of the label relative to the field.
<a href="#">LabelSize</a>	Retrieves or sets a value that indicates the size of the label.
<a href="#">LastError</a>	Retrieves the description for the last error.
<a href="#">Layout</a>	Retrieves or sets a value that indicates the way how fields are arranged.
<a href="#">LayoutHeight</a>	Retrieves a value that indicates the height that's required so all editors fit the control's client area.
<a href="#">LayoutWidth</a>	Retrieves a value that indicates the width that's required so all editors fit the control's client area.
<a href="#">Picture</a>	Retrieves or sets a graphic to be displayed in the control.
<a href="#">PictureDisplay</a>	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background
	Retrieves or sets a value that indicates the index of the

[RadiolImage](#) image for the radio button fields.

[Refresh](#) Refreshes the control.

[Remove](#) Removes an editor.

[RemoveAll](#) Removes all the editors in the control.

[Replacelcon](#) Adds a new icon, replaces an icon or clears the control's image list.

[ScrollBars](#) Specifies the type of scroll bars that control adds.

[SelBackColor](#) Retrieves or sets a value that indicates the selection background color.

[SelfForeColor](#) Retrieves or sets a value that indicates the selection foreground color.

[ShowImageList](#) Specifies whether the control's image list window is visible or hidden.

[ShowToolTip](#) Shows the specified tooltip at given position.

[Template](#) Specifies the control's template.

[TemplateDef](#) Defines inside variables for the next Template/ExecuteTemplate call.

[TemplatePut](#) Defines inside variables for the next Template/ExecuteTemplate call.

[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.

[UseTabKey](#) Retrieves or sets a value that indicates whether the Tab key navigate through the control's fields.

[UseVisualTheme](#) Specifies whether the control uses the current visual theme to display certain UI parts.

[VBorderField](#) Returns or sets a value that indicates the distance between two fields on the vertical axis.

[Version](#) Retrieves the control's version.

[VFieldCount](#) Sets or gets a value that indicates the number of fields on

the vertical axis.

[VisualAppearance](#)

Retrieves the control's appearance.

# method Record.Add (Label as Variant, Type as EditTypeEnum, [Key as Variant])

Adds an editor and returns a reference to the newly created object.

Type	Description
Label as Variant	A string expression that indicates the label of the editor. The Label paramater may include built-in HTML format like described bellow.
Type as <a href="#">EditTypeEnum</a>	An EditTypeEnum expression that indicates the type of the editor being added.
Key as Variant	A Variant expression that indicates the key of the editor being added. If missing, the editor's key is the editor's label.

Return	Description
<a href="#">Editor</a>	An Editor object being created.

The Add method adds a new editor to the control. Use the [DataSource](#) property to bind a recordset to the control. The [Value](#) property indicates the editor's value. Use the [UserEditor](#) method to create an inner ActiveX control, if the Type parameter is UserEditorType. Use the [Label](#) property to get the editor's label. Use the [Item](#) property to access an editor by its key or by its index. Use the [Layout](#) property to arrange fields in the control. Use the [BeginUpdate](#) and [EndUpdate](#) methods to maintain the performance while adding multiple editors. Use the [EnsureVisible](#) method to ensures that an editor fits the control's client area. Use the [EditType](#) property to change the editor's type at runtime. Use the [Remove](#) method to remove an editor. Use the [UserData](#) property to associate an extra data to an editor.

The following VB sample adds an editor to mask a phone number:

```
With Record1
    .BeginUpdate
    With .Add("Phone", EXRECORDLibCtl.MaskType)
        .Mask = "(###) ### - ####"
        .Value = "(245) 282 - 1290"
    End With
    .EndUpdate
End With
```

The following VC sample adds a password editor:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;  
CEditor editor = m_record.Add(COleVariant("Password"), /*EditType*/ 1, vtMissing );  
editor.SetOption( /*exEditPassword*/ 18, COleVariant( (long)TRUE ) );  
editor.SetValue( COleVariant( "pass" ) );
```

The Label parameter may include built-in HTML tags like follows:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" 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</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rr gg bb> ... </fgcolor>** or **<fgcolor=rr gg bb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rr gg bb> ... </bgcolor>** or **<bgcolor=rr gg bb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rr gg bb> ... </solidline>** or **<solidline=rr gg bb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rr gg bb> ... </dotline>** or **<dotline=rr gg bb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the

index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.

- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b>&gt;bold&lt;/b>**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<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 **<font>** 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 "**<font ;18><gra FFFFFFFF;1;1>**gradient-center**</gra></font>**" 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 **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><out 000000>**  
**<fgcolor=FFFFFF>**outlined**</fgcolor></out></font>**" 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 **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><sha>shadow</sha></font>**" generates the following picture:

**shadow**

or "**<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>**" gets:

**outline anti-aliasing**

# property Record.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
<a href="#">AppearanceEnum</a>	An AppearanceEnum expression that indicates the control's appearance.

Use the Appearance property to remove the borders of the control. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color. Use the [Layout](#) property to arrange the fields in the control.

## method Record.AttachTemplate (Template as Variant)

Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.

Type	Description
Template as Variant	A string expression that specifies the Template to execute.

The AttachTemplate/x-script code is a simple way of calling control/object's properties, methods/events using strings. The AttachTemplate features allows you to attach a x-script code to the component. The AttachTemplate method executes x-script code ( including events ), from a string, file or a safe array of bytes. This feature allows you to run any x-script code for any configuration of the component /COM, /NET or /WPF. Exontrol owns the x-script implementation in its easiest form and it does not require any VB engine or whatever to get executed. The x-script code can be converted to several programming languages using the eXHelper tool.

The following sample opens the Windows Internet Explorer once the user clicks the control ( /COM version ):

```
AttachTemplate("handle Click(){ CreateObject(`internetexplorer.application`){ Visible = True; Navigate(`https://www.exontrol.com`) } } ")
```

This script is equivalent with the following VB code:

```
Private Sub Record1_Click()  
    With CreateObject("internetexplorer.application")  
        .Visible = True  
        .Navigate ("https://www.exontrol.com")  
    End With  
End Sub
```

The AttachTemplate/x-script syntax in BNF notation is defined like follows:

```
<x-script> := <lines>  
<lines> := <line>[<eol> <lines>] | <block>  
<block> := <call> [<eol>] { [<eol>] <lines> [<eol>] } [<eol>]  
<eol> := ";" | "\r\n"  
<line> := <dim> | <createobject> | <call> | <set> | <comment> | <handle>[<eol>][<eol>]  
<lines>[<eol>][<eol>]  
<dim> := "DIM" <variables>  
<variables> := <variable> [, <variables>]
```

```

<variable> := "ME" | <identifier>
<createobject> := "CREATEOBJECT(`"<type>`")"
<call> := <variable> | <property> | <variable>."<property>" | <createobject>."<property>"
<property> := [<property>"."]<identifier>["("<parameters>")"]
<set> := <call> "=" <value>
<property> := <identifier> | <identifier> "["<parameters>"]"
<parameters> := <value> [","<parameters>]
<value> := <boolean> | <number> | <color> | <date> | <string> | <createobject> | <call>
<boolean> := "TRUE" | "FALSE"
<number> := "0X"<hexa> | ["-"]<integer>["."<integer>]
<digit10> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit16> := <digit10> | A | B | C | D | E | F
<integer> := <digit10> [<integer>]
<hexa> := <digit16> [<hexa>]
<color> := "RGB("<integer>","<integer>","<integer>")"
<date> := "#"<integer>"/"<integer>"/"<integer>" "["<integer>":"<integer>":"<integer>"]"#
<string> := ""<text>"" | ""<text>""
<comment> := ""<text>
<handle> := "handle " <event>
<event> := <identifier> "["<eparameters>"]"
<eparameters> := <eparameter> [","<eparameters>]
<parameters> := <identifier>

```

where:

<identifier> indicates an identifier of the variable, property, method or event, and should start with a letter.

<type> indicates the type the CreateObject function creates, as a progID for /COM version or the assembly-qualified name of the type to create for /NET or /WPF version

<text> any string of characters

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" ( newline characters ) or ";" character.

The advantage of the AttachTemplate relative to [Template](#) / [ExecuteTemplate](#) is that the AttachTemplate can add handlers to the control events.

# property Record.BackColor as Color

Specifies the control's background color.

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

Use the BackColor property to specify the control's background color. Use the [Picture](#) property to put a picture in the control's background. Use the [BackColor](#) property to change the editor's background color. Use the [LabelBackColor](#) property to specify the background color for the editor's label. Use the [ForeColor](#) property to specify the control's foreground color.

# property Record.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 <a href="#">BackgroundPartEnum</a>	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 <a href="#">Add</a> 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 [BeginUpdate](#) and [EndUpdate](#) methods to maintain performance while init the control. Use the [Refresh](#) method to refresh the control.

## method Record.BeginUpdate ()

Maintains performance when editors are added to the control one at a time.

Type	Description
------	-------------

This method prevents the control from painting until the [EndUpdate](#) method is called. Use the [DataSource](#) property to bind a recordset to the control.

The following VB sample adds 10 EditType editors in the control:

```
With Record1
    .BeginUpdate
    Dim i As Long
    For i = 1 To 10
        .Add "Editor <b>" & i & "</b>", EditType
    Next
    .EndUpdate
End With
```

The following VC sample adds 10 EditType editors in the control:

```
m_record.BeginUpdate();
for ( long i = 1; i < 10; i++ )
{
    COleVariant vtMissing; vtMissing.vt = VT_ERROR;
    CString strLabel;
    strLabel.Format( "Editor <b>%i</b>", i );
    CEditor editor = m_record.Add( COleVariant(strLabel), /*EditType*/ 1, vtMissing );
    editor.SetValue( COleVariant( i ) );
}
m_record.EndUpdate();
```

# property Record.BorderHeight as Long

Sets or retrieves a value that indicates the border height of the control.

Type	Description
Long	A long expression that indicates the height of the control's border.

The BorderHeight property specifies the height of the control's border. By default, the BorderHeight property is 2 pixels. Use the [BorderWidth](#) property to specify the width of the control's border. The control's client area excludes the borders. The fields are arranged in the control's client area. The [HBorderField](#) property specifies a value that indicates the distance between two fields on the horizontal axis. The [VBorderField](#) property specifies a value that indicates the distance between two fields on the vertical axis.



# property Record.BorderWidth as Long

Sets or retrieves a value that indicates the border width of the control.

Type	Description
Long	A long expression that indicates the width of the control's border.

The BorderWidth property specifies the width of the control's border. By default, the BorderWidth property is 2 pixels. Use the [BorderHeight](#) property to specify the height of the control's border. The control's client area excludes the borders. The fields are arranged in the control's client area. The [HBorderField](#) property specifies a value that indicates the distance between two fields on the horizontal axis. The [VBorderField](#) property specifies a value that indicates the distance between two fields on the vertical axis.

# property Record.CheckImage(State as Long) as Long

Retrieves or sets a value that indicates the index of the image for the checkbox fields.

Type	Description
State as Long	A long expression that defines the state of the check box being changed. 0 - unchecked, 1 - checked, 2 - partial checked.
Long	A long expression that indicates the index of the icon used. If the index is not valid the default icon is used.

Use the CheckImage property to change the appearance of the check boxes in the control. Use the [Images](#) method to add a list of icons to the control. Use the [CheckValueType](#) editor to add a check box editor.

Boolean 

The following VB sample changes the appearance for the check boxes:

```
With Record1
    .BeginUpdate
    .Images
" gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

    .CheckImage(0) = 1
    .CheckImage(1) = 2
    With .Add("Boolean", EXRECORDLibCtl.CheckValueType)
        .Option(exCheckValue2) = 1
        .Value = True
    End With
    .EndUpdate
End With
```

The following VC sample changes the appearance for the check boxes:

```
m_record.BeginUpdate();
CString s(
" gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn
);
s = s +
```

"cWnO9B1pZ1Mz/ctpWpc1KV88Ftp5brct43bbXvfT5WfP183/e2AXw3F53pUqptnfqqMvgq

```
m_record.Images(COleVariant( s ));  
m_record.SetCheckImage( 0, 1 );  
m_record.SetCheckImage( 1, 2 );  
COleVariant vtMissing; vtMissing.vt = VT_ERROR;  
CEditor editor = m_record.Add( COleVariant("Boolean"), /*CheckValueType*/ 19, vtMissing  
);  
editor.SetOption( /*exCheckValue2*/ 17, COleVariant( long(1) ) );  
COleVariant vtValue;  
vtValue.vt = VT_BOOL;  
V_BOOL( &vtValue; ) = VARIANT_TRUE;  
editor.SetValue( vtValue );  
m_record.EndUpdate();
```

# property Record.Count as Long

Returns the number of editors in a control.

Type	Description
Long	A long expression that indicates the number editors in the control.

The Count property counts the number of editors in the control. Use the [Add](#) method to add new editors to the control. Use the [DataSource](#) property to bind a recordset to the control. Use the [Item](#) property to access an editor by its index or by its index. Use the [ItemByPosition](#) property to access an editor by its position.

The following VB sample enumerates the visible editors in the control, as they are created:

```
Dim i As Long
With Record1
    For i = 0 To .Count - 1
        Dim e As EXRECORDLibCtl.Editor
        Set e = .Item(i)
        If (e.Visible) Then
            Debug.Print e.Label
        End If
    Next
End With
```

The following VB sample enumerates all editors in the control:

```
Dim e As EXRECORDLibCtl.Editor
For Each e In Record1
    Debug.Print e.Label
Next
```

The following VC sample enumerates all editors in the control:

```
for ( long i = 0; i < m_record.GetCount(); i++ )
{
    CEditor editor = m_record.GetItem( COleVariant( i ) );
    TCHAR szOutput[1024];
    wprintf( szOutput, "%s\n", (LPCTSTR)editor.GetLabel() );
}
```

```
OutputDebugString( szOutput );
```

```
}
```

# method Record.CustomLayout (X as Variant, Y as Variant)

Specifies an array of relative positions that are used when the control arranges the fields on the page.

Type	Description
X as Variant	A safe array of numeric values that indicates the x coordinate, a numeric value that indicates the x position being inserted. A positive value means the absolute position. A negative value means a relative position, For instance, the 100 means that the field will be positioned at 100 pixels from the left side of the control's client area. The -.5 means that the field will be positioned at the center of the control's client area.
Y as Variant	A safe array of numeric values that indicates the y coordinate, a numeric value that indicates the y position being inserted. A positive value means the absolute position. A negative value means a relative position, For instance, the 100 means that the field will be positioned at 100 pixels from the top side of the control's client area. The -.5 means that the field will be positioned at the center of the control's client area.

The CustomLayout method adds new coordinates for arranging the fields when [Layout](#) property is exCustomLayout. The CustomLayout method has effect only if the Layout property is exCustomLayout. Use the CustomLayout property to arrange the fields in a custom order. Use the [Position](#) property to change the position of the editor. Please be aware that calling the Layout property erases all previous position being added by the CustomLayout method. The CustomLayout method must be called after calling the Layout property. Use the [FieldWidth](#) property to specify the width of the fields/editors.

The following VB sample arranges the fields from the left to the right:

```
With Record1
    .BeginUpdate
    .FieldWidth = 96
    .Layout = exLeftToRight
    Dim i As Long
    For i = 1 To 10
        With .Add("Editor <b>" & i & "</b>", EditType)
            .Value = i
        End With
    Next i
End With
```

Editor <b>1</b>	1	Editor <b>2</b>	2
Editor <b>3</b>	3	Editor <b>4</b>	4
Editor <b>5</b>	5	Editor <b>6</b>	6
Editor <b>7</b>	7	Editor <b>8</b>	8
Editor <b>9</b>	9	Editor <b>10</b>	10

Next  
.EndUpdate  
End With

The following VB sample arranges the fields from the top to the bottom:

```
With Record1
    .BeginUpdate
    .FieldWidth = 96
    .Layout = exTopToBottom
    Dim i As Long
    For i = 1 To 10
        With .Add("Editor <b>" & i & "</b>", EditType)
            .Value = i
        End With
    Next
    .EndUpdate
End With
```

Editor <b>1</b>	1	Editor <b>9</b>	9
Editor <b>2</b>	2	Editor <b>10</b>	10
Editor <b>3</b>	3		
Editor <b>4</b>	4		
Editor <b>5</b>	5		
Editor <b>6</b>	6		
Editor <b>7</b>	7		
Editor <b>8</b>	8		

The following VB sample arranges the fields in a circle:

```
Dim n As Long, pi As Double
pi = 3.1415
n = 10
With Record1
    .BeginUpdate
    .FieldWidth = 96
    .Layout = exCustomLayout
    Dim i As Long
    For i = 1 To n
        With .Add("Editor <b>" & i & "</b>", EditType)
            .Value = i
            ' If negative numbers are used, the absolute value represents the coordinate
            ' proportionally with the control's size. In this case the control is consider as being 0..1
            Record1.CustomLayout -(0.5 + Sin(i * 2 * pi / n) / 2), -(0.5 + Cos(i * 2 * pi / n) / 2)
        End With
    Next
    .EndUpdate
End With
```

		Editor <b>5</b>	5		
	Editor <b>6</b>	6		Editor <b>4</b>	4
Editor <b>7</b>	7			Editor <b>3</b>	3
Editor <b>8</b>	8			Editor <b>2</b>	2
		Editor <b>9</b>	9	Editor <b>1</b>	1
		Editor <b>10</b>	10		





# property Record.DataSource as Object

Retrieves or sets a value that indicates the data source for object.

Type	Description
Object	An Object that defines the control's data. Currently, the control supports ADO.Recordset, ADODB.Recordset objects, DAO recordsets

use the DataSource property to bind a recordset to a control. The DataSource property clears the editors collection and add a new editor for each field found in the recordset. The [Key](#) property specifies the field's name. The [Value](#) property is updated as soon as the cursor is moving in the recordset. Use the [Refresh](#) method to update the values for the editors, in case an DAO recordset is used. The [EditType](#) property specifies the type of the editor being inserted. For instance, if the recordset includes a Date/Time field editor, the EditType property is set to DateType. Use the [AddItem](#) or [InsertItem](#) methods to add new predefined values to a drop down list editor. Use the [Item](#) property to access an editor giving its key. Use the [Visible](#) property to hide an editor. Use the [Add](#) method to add new editors to the control. The [Change](#) event is called when the uses changes the value for an editor. If the control is bounded to a recordset, the value of the field is automatically updated in the recodset too. The [LastError](#) property gets the description of the last error, if occurs.

The following VB sample binds the "Employees" table in the "NWIND.MDB" database to the component, using an ADODB recordset:

```
Dim rs As Object, strNWIND As String
strNWIND = "D:\Program Files\Microsoft Visual
Studio\VB98\NWIND.MDB"
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Employees",
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " &
strNWIND, 3, 3 ' Opens the table using static mode
With Record1
.BeginUpdate
Set .DataSource = rs
With .Item("ReportsTo")
.EditType = DropDownListType
Dim i As Long
i = 1
.AddItem 0, "- <b>unspecified</b> -"
```

EmployeeID	1
LastName	Davolio
FirstName	Nancy
Title	Sales Representative
TitleOfCo...	Ms.
BirthDate	12/8/1948
HireDate	5/1/1992
Address	507 - 20th Ave. E. Apt. 2A
City	Seattle
Region	WA
PostalCode	98122
Country	USA
HomePho...	(206) 555-9857
Extension	5467
Photo	Bitmap Image
Notes	Education includes a BA in psychology
ReportsTo	Andrew Fuller

```

While Not rs.EOF
    .AddItem rs("EmployeeID").Value,
rs("FirstName").Value & " <b>" & rs("LastName").Value & "
</b>"
    i = i + 1
    rs.MoveNext
Wend
rs.MoveFirst
End With
.EndUpdate
End With

```

The following VC sample binds the "Employees" table in the "NWIND.MDB" database to the component, using an ADO DB recordset:

```

#import rename ( "EOF", "adoEOF" )
using namespace ADO DB;

BOOL isInstalled(BSTR strProgID, IDispatch** ppObject )
{
    CLSID clsid = CLSID_NULL;
    HRESULT hResult = E_POINTER;
    if (SUCCEEDED( hResult = CLSIDFromProgID( strProgID, &clsid; ) ))
    {
        IDispatch* pObj = NULL;
        if ( SUCCEEDED( hResult = CoCreateInstance( clsid, NULL, CLSCTX_ALL, IID_IDispatch,
reinterpret_cast(&pObj;) ) ) )
        {
            if ( pObj )
                (*ppObject = pObj)->AddRef();
            pObj->Release();
            return TRUE;
        }
    }
    return FALSE;
}

COleVariant vtMissing; vtMissing.vt = VT_ERROR;

```

```
CString strDatabase = "D:\\Program Files\\Microsoft Visual  
Studio\\VB98\\NWIND.MDB";
```

```
CString strError = "";  
IDispatch* pObject = NULL;  
if ( isInstalled( L"ADODB.Recordset", &pObject; ) )  
{  
    _RecordsetPtr spRecordSet;  
    if ( spRecordSet = pObject )  
    {  
        CString strConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source= ");  
        strConnection += strDatabase;  
        try  
        {  
            if ( SUCCEEDED( spRecordSet->Open( v("Employees"), v(strConnection),  
adOpenStatic, adLockOptimistic, 0 ) ) )  
            {  
                CEditor editor;  
  
                m_record.BeginUpdate();  
                m_record.SetLabelSize(96);  
                m_record.SetHBorderField( 6 );  
                m_record.SetDataSource( spRecordSet );  
  
                editor = m_record.GetItem(v("ReportsTo"));  
                editor.SetEditType( /*DropDownListType*/ 3 );  
                editor.AddItem(0, "- unspecified -", vtMissing );  
                while ( !spRecordSet->adoEOF )  
                {  
                    CString strName = V2S( &spRecordSet->Fields->GetItem( v("FirstName") )->  
>Value );  
                    strName += " ";  
                    strName += V2S( &spRecordSet->Fields->GetItem( v("LastName") )->Value  
);  
                    editor.AddItem( spRecordSet->Fields->GetItem( v("EmployeeID") )->Value,  
strName, vtMissing );  
                    spRecordSet->MoveNext();  
                }  
            }  
        }  
    }  
}
```

```
    }
    spRecordSet->MoveFirst();
    m_record.EndUpdate();
}
}
catch (...)
{
    strError = "The sample database is missing. The 'SAMPLE.MDB' file is not found,
or doesn't contain an 'Employees' table.";
};
}
pObject->Release();
}
else
    strError = "Microsoft ADO DB namespace, 'MSADO15.DLL' file is not installed. ";
```

## property Record.EditorFromPoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS) as Editor

Retrieves the editor from point.

Type	Description
X as OLE_XPOS_PIXELS	A long expression that indicates the X position where the editor is located.
Y as OLE_YPOS_PIXELS	A long expression that indicates the Y position where the editor is located.
<a href="#">Editor</a>	An Editor object that's found at the specified position.

use the EditorFromPoint property to get the editor from the point. Use the [Focus](#) property to get the focused editor. Use the [EnsureVisible](#) method to ensures that an editor fits the control's client area.

The following VB sample prints the editor from the point:

```
Private Sub Record1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim e As EXRECORDLibCtl.Editor
    Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not e Is Nothing Then
        Debug.Print e.Label & " = " & e.Value
    End If
End Sub
```

The following VC sample prints the editor from the point:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
}
```

```
return szDefault;
```

```
}
```

# property Record.Enabled as Boolean

Enables or disables the control.

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

Use the Enabled property to enable or disable the control. Use the [ForeColor](#) property to change the control's foreground color. Use the [Locked](#) property to lock a specified editor. By default, the Enabled property is True.

## method Record.EndUpdate ()

Resumes painting the control after painting is suspended by the BeginUpdate method.

### Type

### Description

Use the BeginUpdate method to maintain performance when editors are added to the control one at a time. Use the [DataSource](#) property to bind a recordset to the control.

The following VB sample adds 10 EditType editors in the control:

```
With Record1
    .BeginUpdate
    Dim i As Long
    For i = 1 To 10
        .Add "Editor <b>" & i & "</b>", EditType
    Next
    .EndUpdate
End With
```

The following VC sample adds 10 EditType editors in the control:

```
m_record.BeginUpdate();
for ( long i = 1; i < 10; i++ )
{
    COleVariant vtMissing; vtMissing.vt = VT_ERROR;
    CString strLabel;
    strLabel.Format( "Editor <b>%i</b>", i );
    CEditor editor = m_record.Add( COleVariant(strLabel), /*EditType*/ 1, vtMissing );
    editor.SetValue( COleVariant( i ) );
}
m_record.EndUpdate();
```



# method Record.EnsureVisible (Index as Variant)

Ensures that a field fits the control's client area.

Type	Description
Index as Variant	A long expression that indicates the index of the editor being requested, a string expression that indicates the key of the editor being accessed.

Use the EnsureVisible method to ensure that a specified editor fits the control's client area. The [Index](#) property specifies the index of the editor. The [Key](#) property specifies the key of the editor. The [ScrollBars](#) property specifies whether the control adds scroll bars when required. The EnsureVisible method scrolls the control's content if is is necessary. If the control has no scroll bars, the EnsureVisible method has no effect.

# method Record.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed
Return	Description
Variant	A Variant expression that indicates the result after executing the Template.

Use the ExecuteTemplate property to returns the result of executing a template file. Use the [Template](#) property to execute a template without returning any result. Use the ExecuteTemplate property to execute code by passing instructions as a string ( template string ).

For instance, the following sample retrieves the number of editors in the data control:

```
Debug.Print Record1.ExecuteTemplate("Count")
```

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence ( when Apply button is pressed ), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string ( template string ).

The Template script is composed by lines of instructions. Instructions are separated by "\n\r" ( newline ) characters.

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. ( Sample: Dim h, h1, h2 )*
- variable = property( list of arguments ) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. ( Sample: h = InsertItem(0,"New Child") )*
- property( list of arguments ) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method( list of arguments ) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- { *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- } *Ending the object's context*
- object. property( list of arguments ).property( list of arguments ).... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

# property Record.FieldHeight as Long

Retrieves or sets a value that indicates the height of the field.

Type	Description
Long	A long expression that specifies the height of the fields.

The FieldHeight property specifies the height of the fields, in pixels. By default, the FieldHeight property is -1. If the FieldHeight property is negative, the field's height is determined by the control's font. Use the [Font](#) property to specify the control's font. Use the [FieldWidth](#) property to specify the width of the fields. Use the [VBorderField](#) property to specify the distance between two fields on the vertical axis. Use the [BorderHeight](#) property to specify the control's border. Use the [VFieldCount](#) property to specify the number of fields on the vertical axis.

# property Record.FieldWidth as Long

Retrieves or sets a value that indicates the width of the field.

Type	Description
Long	A long expression that specifies the width of the fields.

The FieldWidth property specifies the width of the fields, in pixels. By default, the FieldWidth property is -1. If the FieldWidth property is negative, the field's width is the width of the control's client area. The control's client area excludes the control's border. Use the [Font](#) property to specify the control's font. Use the [FieldHeight](#) property to specify the height of the fields. Use the [HBorderField](#) property to specify the distance between two fields on the horizontal axis. Use the [BorderWidth](#) property to specify the control's border. Use the [HFieldCount](#) property to specify the number of fields on the horizontal axis. Use the [LabelSize](#) property to specify the width of the label.

# property Record.Focus as Editor

Specifies the editor that gets the focus.

Type	Description
<a href="#">Editor</a>	An editor that gets the focus.

Use the Focus property to get the editor that has the focus.

# property Record.Font as IFontDisp

Retrieves or sets the control's font.

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

Use the Font property to specify the control's font. Use the [FieldHeight](#) property to specify the height of the fields. The Font property updates the height of the fields, if the FieldHeight property is less than zero. Use the built-in HTML tags like <b>, <u>, <i>, <s> to change the font attributes for parts of the editor's [Label](#).

# property Record.ForeColor as Color

Specifies the control's foreground color.

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

Use the ForeColor property to specify the control's foreground color. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to change the editor's foreground color. Use the [LabelForeColor](#) property to specify the foreground color for the editor's label.



# property Record.HBorderField as Long

Returns or sets a value that indicates the distance between two fields on the horizontal axis.

Type	Description
Long	A long expression that indicates the distance between two fields on the horizontal axis.

The HBorderField property specifies a value that indicates the distance between two fields on the horizontal axis. The [BorderWidth](#) property specifies the width of the control's border. By default, the HBorderField property is 2 pixels. Use the [BorderHeight](#) property to specify the height of the control's border. The control's client area excludes the borders. The fields are arranged in the control's client area. The [VBorderField](#) property specifies a value that indicates the distance between two fields on the vertical axis

# property Record.HFieldCount as Long

Sets or gets a value that indicates the number of fields on the horizontal axis.

Type	Description
Long	A long expression that indicates the number of fields on the horizontal axis.

Use the HFieldCount property to specify the number of fields on the horizontal axis. By default, the HFieldCount property is -1. If the HFieldCount property is -1, the control puts the fields as much as they fit the control's client area. The HFieldCount property has effect only if the [Layout](#) property is exLeftToRight or exTopToBottom. The HFieldCount property has no effect if the Layout property is exCustomLayout. Use the [FieldWidth](#) property to specify the width of the fields. Use the [HBorderField](#) property to specify the distance between two fields on the horizontal axis. Use [VFieldCount](#) property to specify the number of fields on the vertical axis.

# property Record.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	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none"><li>• a string expression that indicates the path to the picture file, being loaded.</li><li>• a string expression that indicates the base64 encoded string that holds a picture object, Use the <a href="#">eximages</a> tool to save your picture as base64 encoded format.</li><li>• A Picture object that indicates the picture being added or replaced. ( A Picture object implements IPicture interface ),</li></ul> <p>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</p>

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 ).

The following sample shows how to put a custom size picture in the column's header:

```
<CONTROL>.HTMLPicture("pic1") = "c:/temp/editors.gif"
<CONTROL>.HTMLPicture("pic2") = "c:/temp/editpaste.gif"

<COLUMN1>.HTMLCaption = "A <img>pic1</img>"
<COLUMN2>.HTMLCaption = "B <img>pic2</img>"
<COLUMN3>.HTMLCaption = "A <img>pic1</img> + B <img>pic2</img>"
```

# property Record.hWnd as Long

Retrieves the control's window handle.

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

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 Record.Images (Handle as Variant)

Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.

Type	Description
------	-------------

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 lVal 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(

Handle as Variant

(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 control provides an image list window, that's displayed at design time. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. Use the [ShowImageList](#) property to hide the image list window, at design time. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list window. At runtime, the user can use the Images and [Replacelcon](#) method to change the Images collection. The Images collection is 1 based. The [Image](#) property assigns an icon to an editor.

Boolean 

The following VB sample changes the appearance for the check boxes:

```
With Record1
    .BeginUpdate
    .Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm

    .CheckImage(0) = 1
    .CheckImage(1) = 2
    With .Add("Boolean", EXRECORDLibCtl.CheckValueType)
        .Option(exCheckValue2) = 1
        .Value = True
    End With
    .EndUpdate
End With
```

The following VC sample changes the appearance for the check boxes:

```
m_record.BeginUpdate();
CString s(
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm
");
s = s +
"cWnO9B1pZ1Mz/ctpWpc1KV88Ftp5brct43bbXvfT5WfP183/e2AXw3F53pUqptnfqqMvgq"
```

```
m_record.Images(COleVariant( s ));
m_record.SetCheckImage( 0, 1 );
m_record.SetCheckImage( 1, 2 );
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_record.Add( COleVariant("Boolean"), /*CheckValueType*/ 19, vtMissing
);
editor.SetOption( /*exCheckValue2*/ 17, COleVariant( long(1) ) );
COleVariant vtValue;
vtValue.vt = VT_BOOL;
V_BOOL( &vtValue ) = VARIANT_TRUE;
editor.SetValue( vtValue );
m_record.EndUpdate();
```

# property Record.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 Record.item (Index as Variant) as Editor

Returns an editor based on its index.

Type	Description
Index as Variant	A long expression that indicates the index of the editor being requested, or a string expression that indicates the key of the editor being requested.
<a href="#">Editor</a>	An Editor object being accessed.

Use the Item property to access an editor by index or by key. Use the [Index](#) property to retrieve the index of the editor in the control's collection of [Editor](#) objects. Use the [Key](#) property to identify an editor. Use the [Position](#) property to specify the editor's position. Use the [Visible](#) property to hide an editor. By default, the first editor added has the Index property on 0. The Index property of the editor is updated as soon as an editor is removed. Use the [ItemByPosition](#) property to access an editor giving its position.

The following VB sample enumerates the visible editors in the control, as they are created:

```
Dim i As Long
With Record1
    For i = 0 To .Count - 1
        Dim e As EXRECORDLibCtl.Editor
        Set e = .Item(i)
        If (e.Visible) Then
            Debug.Print e.Label
        End If
    Next
End With
```

The following VB sample enumerates all editors in the control:

```
Dim e As EXRECORDLibCtl.Editor
For Each e In Record1
    Debug.Print e.Label
Next
```

The following VC sample enumerates all editors in the control:

```
for ( long i = 0; i < m_record.GetCount(); i++ )
{
```

```
CEditor editor = m_record.GetItem( COleVariant( i ) );  
TCHAR szOutput[1024];  
wsprintf( szOutput, "%s\n", (LPCTSTR)editor.GetLabel() );  
OutputDebugString( szOutput );  
}
```

# property Record.ItemByPosition (Position as Variant) as Editor

Returns an editor based on its position.

Type	Description
Position as Variant	A long expression that indicates the position of the editor being requested.
<a href="#">Editor</a>	An Editor object being requested.

Use the ItemByPosition property to access an editor giving its position. Use the [Position](#) property to specify the editor's position. Use the [Item](#) property to access an editor by index or by key. Use the [Index](#) property to retrieve the index of the editor in the control's collection of [Editor](#) objects. Use the [Key](#) property to identify an editor. Use the [Visible](#) property to hide an editor. By default, the first editor added has the Index property on 0. The Index property of the editor is updated as soon as an editor is removed.

The following VB sample enumerates the visible editors in the control, as they are displayed:

```
Dim i As Long
With Record1
    For i = 0 To .Count - 1
        Dim e As EXRECORDLibCtl.Editor
        Set e = .ItemByPosition(i)
        If (e.Visible) Then
            Debug.Print e.Label
        End If
    Next
End With
```

The following VC sample enumerates the visible editors in the control, as they are displayed:

```
for ( long i = 0; i < m_record.GetCount(); i++ )
{
    CEditor editor = m_record.GetItemByPosition( COleVariant( i ) );
    if ( editor.GetVisible() )
    {
        TCHAR szOutput[1024];
        wsprintf( szOutput, "%s\n", (LPCTSTR)editor.GetLabel() );
    }
}
```

```
OutputDebugString( szOutput );
```

```
}
```

```
}
```

# property Record.LabelAlignment as AlignmentEnum

Specifies the alignment of the label relative to the field.

Type	Description
<a href="#">AlignmentEnum</a>	An AlignmentEnum expression that indicates the alignment of the label relative to their editors.

Use the LabelAlignment property to right align the labels. Use the [LabelAlignment](#) property to align the label for a specified editor.

# property Record.LabelSize as Long

Retrieves or sets a value that indicates the size of the label.

Type	Description
Long	A long expression that indicates the size of the label.

The LabelSize property specifies the size of the label in the editors. By default, the LabelSize property is 64 pixels. Use the [Label](#) property to specify the editor's label. Use the [FieldWidth](#) property to specify the width of the fields. A field contains the label and the editor. Use the [HBorderField](#) property to specify the distance between two fields on the horizontal axis. Use the [BorderWidth](#) property to specify the control's border.

# property Record.LastError as String

Retrieves the description for the last error.

Type	Description
String	A string expression that indicates the description of the last error.

The LastError property gets the description of the last database error that occurs. Use the [DataSource](#) property to bind a recordset to the control. For instance, the LastError property could get "The field is too small to accept the amount of data you attempted to add. Try inserting or pasting less data", if you are trying to type more characters in a bounded field.

# property Record.Layout as LayoutEnum

Retrieves or sets a value that indicates the way how fields are arranged.

Type	Description
<a href="#">LayoutEnum</a>	A LayoutEnum expression that indicates how the fields are arranged in the control's client area.

The Layout property specifies how the fields are arranged in the control's client area. By default, the Layout property is exLeftToRight. Use the [CustomLayout](#) method to add new positions for the fields, when the Layout property is exCustomLayout. Use the [FieldWidth](#) property to specify the width of the fields. Use the [FieldHeight](#) property to specify the height of the fields. Use the [VFieldCount](#) property to specify the number of fields on the vertical axis. Use [HFieldCount](#) property to specify the number of fields on the horizontal axis.



# property Record.LayoutHeight as Long

Retrieves a value that indicates the height that's required so all editors fit the control's client area.

Type	Description
Long	A long expression that specifies the height that's required so all editors fit the control's client area.

The LayoutHeight property specifies the height, in pixels that's required so all editors fit the control's client area. Use the [Layout](#) property to arrange the fields in the page. Use the [LayoutWidth](#) property specifies the width, in pixels that's required so all editors fit the control's client area.

# property Record.LayoutWidth as Long

Retrieves a value that indicates the width that's required so all editors fit the control's client area.

Type	Description
Long	A long expression that specifies the height that's required so all editors fit the control's client area.

Use the LayoutWidth property specifies the width, in pixels that's required so all editors fit the control's client area. Use the [Layout](#) property to arrange the fields in the page. The [LayoutHeight](#) property specifies the height, in pixels that's required so all editors fit the control's client area

# property Record.Picture as IPictureDisp

Retrieves or sets a graphic to be displayed in the control.

Type	Description
IPictureDisp	A Picture object being displayed in the control's background.

Use the Picture property to put a picture in the control's background. Use the [PictureDisplay](#) property to specify how the picture is displayed on the control's background. Use the [BackColor](#) property to specify the control's background color. Use the [BackColor](#) property to change the editor's background color. Use the [LabelBackColor](#) property to specify the background color for the editor's label. Use the [Picture](#) property to assign a custom size to an editor.

# property Record.PictureDisplay as PictureDisplayEnum

Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background

Type	Description
<a href="#">PictureDisplayEnum</a>	A PictureDisplayEnum expression that specifies how the picture is displayed on the control's background.

Use the PictureDisplay property to specify how the picture is displayed on the control's background. Use the [Picture](#) property to put a picture in the control's background. Use the [BackColor](#) property to specify the control's background color. Use the [BackColor](#) property to change the editor's background color. Use the [LabelBackColor](#) property to specify the background color for the editor's label. Use the [Picture](#) property to assign a custom size to an editor.

# property Record.RadiolImage(Checked as Boolean) as Long

Retrieves or sets a value that indicates the index of the image for the radio button fields.

Type	Description
Checked as Boolean	A boolean expression that specifies whether the image is requested for checked/un-checked items
Long	A long expression that specifies the index of the icon to display

# method Record.Refresh ()

Refreshes the control.

Type	Description
------	-------------

Use the Refresh method to update the values for all editors. The [BeginUpdate](#) method maintains performance when editors are added to the control one at a time. The [EndUpdate](#) method resumes painting the control after painting is suspended by the BeginUpdate method.

# method Record.Remove (Index as Variant)

Removes an editor.

Type	Description
Index as Variant	A long expression that indicates the index of the editor being removed, a string expression that indicates the key of the editor being removed.

Use the Remove method to remove an editor. Use the [RemoveAll](#) method to remove all editors in the control. Use the [Add](#) method to add new editors to the control. Use the [RemoveItem](#) property to remove a predefined value from a drop down list editor. Use the [RemoveButton](#) method to remove a button from an editor.

# method Record.RemoveAll ()

Removes all the editors in the control.

Type	Description
------	-------------

Use the RemoveAll method to remove all editors in the control. Use the [Remove](#) method to remove an editor. Use the [Add](#) method to add new editors to the control. Use the [ClearItems](#) method to remove all predefined values from a drop down list editor. Use the [ClearButtons](#) method to remove all buttons in the editor.



# method Record.Replacelcon ([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	<p>A Variant expression that specifies the icon to add or insert, as one of the following options:</p> <ul style="list-style-type: none"><li>• a long expression that specifies the handle of the icon (HICON)</li><li>• a string expression that indicates the path to the picture file</li><li>• a string expression that defines the picture's content encoded as BASE64 strings using the <a href="#">eXImages</a> tool</li><li>• 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)</li></ul> <p>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.</p> <p>By default, if the Icon parameter is not specified or is missing, a value of 0 is used.</p>
Index as Variant	<p>A long expression that defines the index of the icon to insert or remove, as follows:</p> <ul style="list-style-type: none"><li>• 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)</li><li>• A negative value clears all icons when the Icon parameter is zero</li></ul> <p>By default, if the Index parameter is not specified or is missing, a value of -1 is used.</p>
Return	Description
Long	A long expression that indicates the index of the icon in the images collection

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

collection. Also, the `Replacelcon` 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 = Record1.Replacelcon( 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 = Record1.Replacelcon( 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:

`Record1.Replacelcon 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:

`Record1.Replacelcon 0, -1`

# property Record.ScrollBars as ScrollBarsEnum

Specifies the type of scroll bars that control adds.

Type	Description
<a href="#">ScrollBarsEnum</a>	A ScrollBarsEnum expression that indicates which scroll bars will be visible in the control.

Use the ScrollBars property to specify what scroll bars the control adds. By default, the control adds scroll bars when they are required. Use the [LayoutWidth](#) property to get the width in pixels that's required so no horizontal scroll bar is required. Use the [LayoutHeight](#) property to get the height in pixels that's required so no vertical scroll bar is required.

# property Record.SelBackColor as Color

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

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

Use the SelBackColor property to specify the selection background color. Use the [SelForeColor](#) property to specify the selection foreground color. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color.

# property Record.SelForeColor as Color

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

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

Use the SelForeColor property to specify the selection foreground color. Use the [SelBackColor](#) property to specify the selection background color. Use the [ForeColor](#) property to specify the control's foreground color. Use the [BackColor](#) property to specify the control's background color.

# property Record.ShowImageList as Boolean

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

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

The property has effect only at design time. Use the [Images](#) method to assign a list of icons at runtime. Use the [Replacelcon](#) method to update the control's list of icons. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list panel.

**method Record.ShowToolTip (ToolTip as String, [Title as Variant], [Alignment as Variant], [X as Variant], [Y as Variant])**

Shows the specified tooltip at given position.

Type	Description
ToolTip as String	<p>The ToolTip parameter can be any of the following:</p> <ul style="list-style-type: none"><li>• NULL(BSTR) or "&lt;null&gt;"(string) to indicate that the tooltip for the object being hovered is not changed</li><li>• A String expression that indicates the description of the tooltip, that supports built-in HTML format (adds, replaces or changes the object's tooltip)</li></ul>
Title as Variant	<p>The Title parameter can be any of the following:</p> <ul style="list-style-type: none"><li>• missing (VT_EMPTY, VT_ERROR type) or "&lt;null&gt;" (string) the title for the object being hovered is not changed.</li><li>• A String expression that indicates the title of the tooltip (no built-in HTML format) (adds, replaces or changes the object's title)</li></ul>
Alignment as Variant	<p>A long expression that indicates the alignment of the tooltip relative to the position of the cursor. If missing (VT_EMPTY, VT_ERROR) the alignment of the tooltip for the object being hovered is not changed.</p> <p>The Alignment parameter can be one of the following:</p> <ul style="list-style-type: none"><li>• 0 - exTopLeft</li><li>• 1 - exTopRight</li><li>• 2 - exBottomLeft</li><li>• 3 - exBottomRight</li><li>• 0x10 - exCenter</li><li>• 0x11 - exCenterLeft</li><li>• 0x12 - exCenterRight</li><li>• 0x13 - exCenterTop</li><li>• 0x14 - exCenterBottom</li></ul> <p>By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).</p>

Specifies the horizontal position to display the tooltip as one of the following:

- missing (VT\_EMPTY, VT\_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current horizontal position of the cursor (current x-position)
- a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position)
- a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)

X as Variant

---

Specifies the vertical position to display the tooltip as one of the following:

- missing (VT\_EMPTY, VT\_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current vertical position of the cursor (current y-position)
- a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position)
- a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Y as Variant

---

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the [MouseMove](#) event. Use 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 [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to change the tooltip's font. 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.

For instance:

- [ShowToolTip\(<null>, <null>, , +8, +8\)](#), shows the tooltip of the object moved relative



to its default position

- `ShowToolTip(<null>`,`new title`)`, adds, changes or replaces the title of the object's tooltip
- `ShowToolTip(`new content`)`, adds, changes or replaces the object's tooltip
- `ShowToolTip(`new content`,`new title`)`, shows the tooltip and title at current position
- `ShowToolTip(`new content`,`new title`,`+8`,`+8`)`, shows the tooltip and title moved relative to the current position
- `ShowToolTip(`new content`,``,`128,128`)`, displays the tooltip at a fixed position
- `ShowToolTip(``,``)`, hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

- `<b> ... </b>` displays the text in **bold**
- `<i> ... </i>` displays the text in *italics*
- `<u> ... </u>` underlines the text
- `<s> ... </s>` Strike-through text
- `<a id;options> ... </a>` displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The `<a>` element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- `<font face;size> ... </font>` displays portions of text with a different font and/or different size. For instance, the "`<font Tahoma;12>bit</font>`" 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</font>`" displays the bit text using the current font, but with a different size.
- `<fgcolor rrggbb> ... </fgcolor>` or `<fgcolor=rrggbb> ... </fgcolor>` displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<bgcolor rrggbb> ... </bgcolor>` or `<bgcolor=rrggbb> ... </bgcolor>` displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<solidline rrggbb> ... </solidline>` or `<solidline=rrggbb> ... </solidline>` draws a solid-line on the bottom side of the current text-line, of specified RGB color. The `<solidline> ... </solidline>` draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<dotline rrggbb> ... </dotline>` or `<dotline=rrggbb> ... </dotline>` draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<upline> ... </upline>` draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).

- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. 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 part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;** ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. 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 **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<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 **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>subscript**" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>superscript**" displays the text such as: Text with superscript
- **<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 **<font>** 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 "**<font ;18><gra FFFFFFFF;1;1>gradient-center</gra></font>**" 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 **<font>** HTML tag can be used to define the

height of the font. For instance the "<font ;31><out 000000>

<fgcolor=FFFFFF>outlined</fgcolor></out></font>" 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 <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

or "<font ;31><sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>" gets:

outline anti-aliasing

# property Record.Template as String

Specifies the control's template.

Type	Description
String	A string expression that defines the control's template.

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string ( template string ). Use the [ExecuteTemplate](#) property to execute a template script and gets the result.

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence ( when Apply button is pressed ), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string ( template string ).

The Template script is composed by lines of instructions. Instructions are separated by "\n\r" ( newline ) characters.

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. ( Sample: Dim h, h1, h2 )*
- variable = property( list of arguments ) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values*

*separated by commas. ( Sample: `h = InsertItem(0,"New Child")` )*

- *property( list of arguments ) = value* *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- *method( list of arguments )* *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- *{* *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- *}* *Ending the object's context*
- *object. property( list of arguments ).property( list of arguments )....* *The .(dot) character splits the object from its property. For instance, the `Columns.Add("Column1").HeaderBackColor = RGB(255,0,0)`, adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: `BackColor = RGB(255,0,0)`*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

# property Record.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus** or **XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

```
with (Control)
    TemplateDef = [Dim var_Column]
    TemplateDef = var_Column
    Template = [var_Column.Def(4) = 255]
endwith
```

This sample allocates a variable var\_Column, assigns the value to the variable ( the second call of the TemplateDef ), and the Template call uses the var\_Column variable ( as an object ), to call its Def property with the parameter 4.

Let's say we need to define the background color for a specified column, so we need to call the Def(exCellBackColor) property of the column, to define the color for all cells in the column.

The following **VB6** sample shows setting the Def property such as:

```
With Control
    .Columns.Add("Column 1").Def(exCellBackColor) = 255
    .Columns.Add "Column 2"
    .Items.AddItem 0
    .Items.AddItem 1
```

```
.Items.AddItem 2  
End With
```

In **dBASE Plus**, calling the Def(4) has no effect, instead using the TemplateDef helps you to use properly the Def property as follows:

```
local Control,var_Column  
  
Control = form.Active1.nativeObject  
// Control.Columns.Add("Column 1").Def(4) = 255  
var_Column = Control.Columns.Add("Column 1")  
with (Control)  
    TemplateDef = [Dim var_Column]  
    TemplateDef = var_Column  
    Template = [var_Column.Def(4) = 255]  
endwith  
Control.Columns.Add("Column 2")  
Control.Items.AddItem(0)  
Control.Items.AddItem(1)  
Control.Items.AddItem(2)
```

The equivalent sample for **XBasic in A5**, is as follows:

```
Dim Control as P  
Dim var_Column as P  
  
Control = topparent:CONTROL_ACTIVEX1.activex  
' Control.Columns.Add("Column 1").Def(4) = 255  
var_Column = Control.Columns.Add("Column 1")  
Control.TemplateDef = "Dim var_Column"  
Control.TemplateDef = var_Column  
Control.Template = "var_Column.Def(4) = 255"  
  
Control.Columns.Add("Column 2")  
Control.Items.AddItem(0)  
Control.Items.AddItem(1)  
Control.Items.AddItem(2)
```

The samples just call the `Column.Def(4) = Value`, using the `TemplateDef`. The first call of `TemplateDef` property is `"Dim var_Column"`, which indicates that the next call of the `TemplateDef` will defines the value of the variable `var_Column`, in other words, it defines the object `var_Column`. The last call of the `Template` property uses the `var_Column` member to use the x-script and so to set the `Def` property so a new color is being assigned to the column.

The `TemplateDef`, [Template](#) and [ExecuteTemplate](#) support x-script language ( `Template` script of the `Exontrols` ), like explained bellow:

The `Template` or x-script is composed by lines of instructions. Instructions are separated by `"\n\r"` ( newline characters ) or `";"` character. The `;` character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas.* ( Sample: `Dim h, h1, h2` )
- `variable = property( list of arguments )` *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas.* ( Sample: `h = InsertItem(0,"New Child")` )
- `property( list of arguments ) = value` *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- `method( list of arguments )` *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- `{` *Beginning the object's context. The properties or methods called between `{` and `}` are related to the last object returned by the property prior to `{` declaration.*
- `}` *Ending the object's context*
- `object.property( list of arguments ).property( list of arguments )....` *The `.` (dot) character splits the object from its property. For instance, the `Columns.Add("Column1").HeaderBackColor = RGB(255,0,0)`, adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with `0x` which indicates a hexa decimal representation, else it should starts with digit, or `+/-` followed by a digit, and `.` is the decimal separator. Sample: `13` indicates the integer `13`, or `12.45` indicates the double expression `12,45`
- *date* expression is delimited by `#` character in the format `#mm/dd/yyyy hh:mm:ss#`. Sample: `#31/12/1971#` indicates the December 31, 1971
- *string* expression is delimited by `"` or ``` characters. If using the ``` character, please



make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(file)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

## method Record.TemplatePut (newVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
newVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / [TemplateDef](#) property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus** or **XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

The [TemplateDef](#), TemplatePut, [Template](#) and [ExecuteTemplate](#) support x-script language ( Template script of the Exontrols ), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" ( newline characters ) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. ( Sample: Dim h, h1, h2 )*
- variable = property( list of arguments ) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. ( Sample: h = InsertItem(0,"New Child") )*
- property( list of arguments ) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method( list of arguments ) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- { *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- } *Ending the object's context*
- object. property( list of arguments ).property( list of arguments ).... *The .(dot) character splits the object from its property. For instance, the*

*Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The x-script may use constant expressions as follows:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may start with 0x which indicates a hexa decimal representation, else it should start with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by # character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by " or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also, the template or x-script code may support general functions as follows:

- **Me** property indicates the original object.
- **RGB(R,G,B)** property retrieves an RGB value, where the R, G, B are byte values that indicate the R G B values for the color being specified. For instance, the following code changes the control's background color to red: *BackColor = RGB(255,0,0)*
- **LoadPicture(file)** property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.
- **CreateObject(progID)** property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.

# property Record.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.

# property Record.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.

# property Record.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. Use the [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears.

# property Record.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.

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.

# property Record.UseTabKey as Boolean

Retrieves or sets a value that indicates whether the Tab key navigates through the control's fields.

Type	Description
Boolean	A Boolean expression that indicates whether the Tab key navigates through the control's fields.

By default, the UseTabKey property is True. If the user presses the Tab key, while the UseTabKey property is True, the focus is moved to the next visible editor. If the user presses the SHIFT + Tab key, while the UseTabKey property is True, the focus is moved to the previous visible editor. If the UseTabKey property is False, and user presses the Tab key the control loses the focus, and the next visible control in the form gets the focus.



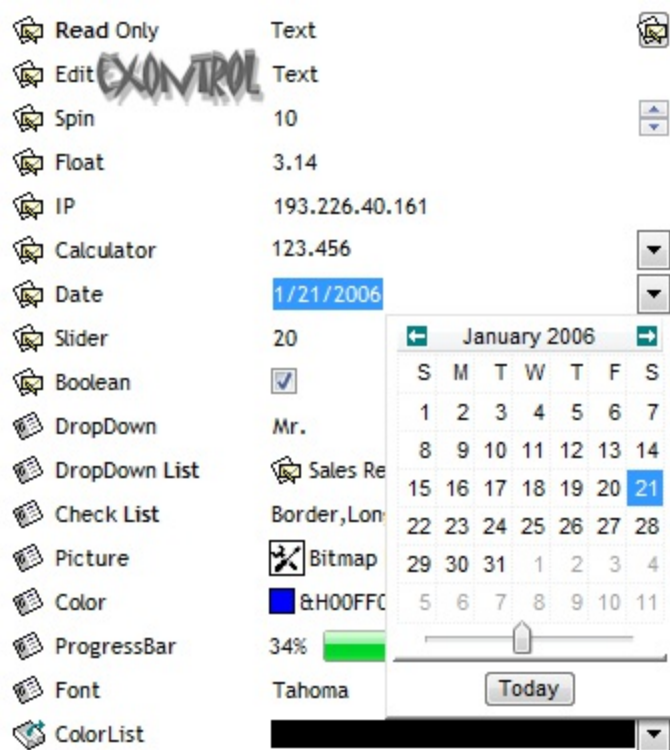
# property Record.UseVisualTheme as UIVisualThemeEnum

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

Type	Description
<a href="#">UIVisualThemeEnum</a>	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 header, filterbar, 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.

The following screen shot shows the control while the UseVisualTheme property is exDefaultVisualTheme:



since the second screen shot shows the same data as the UseVisualTheme property is exNoVisualTheme:

The screenshot shows the Visual Studio IDE with a list of controls on the left and a calendar control on the right. The controls list includes:

- Read Only
- Edit **EXONTROL**
- Spin
- Float
- IP
- Calculator
- Date
- Slider
- Boolean
- DropDown
- DropDown List
- Check List
- Picture
- Color
- ProgressBar
- Font
- ColorList

The calendar control on the right shows the month of January 2006. The date 1/21/2006 is highlighted in blue. The calendar also displays the days of the week (S, M, T, W, T, F, S) and the years 2005 and 2006.

# property Record.VBorderField as Long

Returns or sets a value that indicates the distance between two fields on the vertical axis.

Type	Description
Long	A long expression that indicates the distance in pixels between two fields on the vertical axis.

The VBorderField property specifies a value that indicates the distance between two fields on the vertical axis. The [BorderWidth](#) property specifies the width of the control's border. By default, the VBorderField property is 2 pixels. Use the [BorderHeight](#) property to specify the height of the control's border. The control's client area excludes the borders. The fields are arranged in the control's client area. The [HBorderField](#) property specifies a value that indicates the distance between two fields on the horizontal axis.

# property Record.Version as String

Retrieves the control's version.

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

The Version property specifies the version number of the control you have installed.

# property Record.VFieldCount as Long

Sets or gets a value that indicates the number of fields on the vertical axis, when the control displays a vertical scroll bar.

Type	Description
Long	A long expression that indicates the number of fields on the horizontal axis.

Use the VFieldCount property to specify the number of fields on the vertical axis. By default, the VFieldCount property is -1. If the VFieldCount property is -1, the control puts the fields as much as they fit the control's client area. The VFieldCount property has effect only if the [Layout](#) property is exLeftToRight or exTopToBottom. The VFieldCount property has no effect if the Layout property is exCustomLayout. Use the [FieldHeight](#) property to specify the height of the fields. Use the [VBorderField](#) property to specify the distance between two fields on the vertical axis. Use [HFieldCount](#) property to specify the number of fields on the horizontal axis.

# property Record.VisualAppearance as Appearance

Retrieves the control's appearance.

Type	Description
<a href="#">Appearance</a>	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

# ExRecord 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: {656D66AF-1E46-45E3-B1B5-FFE9FB353AC7}. The object's program identifier is: "Exontrol.Record". The /COM object module is: "ExRecord.dll"

The ExRecord component supports the following events:

Name	Description
<a href="#">ButtonClick</a>	Occurs when the user clicks the editor's button.
<a href="#">Change</a>	Occurs when the user changes the editor's content.
<a href="#">Click</a>	Occurs when the user presses and then releases the left mouse button over the control.
<a href="#">DbClick</a>	Occurs when the user dblclk the left mouse button over an object.
<a href="#">KeyDown</a>	Occurs when the user presses a key while an object has the focus.
<a href="#">KeyPress</a>	Occurs when the user presses and releases an ANSI key.
<a href="#">KeyUp</a>	Occurs when the user releases a key while an object has the focus.
<a href="#">MouseDown</a>	Occurs when the user presses a mouse button.
<a href="#">MouseMove</a>	Occurs when the user moves the mouse.
<a href="#">MouseUp</a>	Occurs when the user releases a mouse button.
<a href="#">RClick</a>	Fired when right mouse button is clicked.
<a href="#">UserEditorOleEvent</a>	Occurs when an user editor fires an event.

# event ButtonClick (Ed as Editor, Key as Variant)

Occurs when the user clicks the editor's button.

Type	Description
Ed as <a href="#">Editor</a>	An Editor object where the event occurs.
Key as Variant	A string expression that indicates the key of the button being pressed.

The ButtonClick event notifies your application that the user clicks a button. The [AddButton](#) method inserts new buttons to the editor. Use the [RemoveButton](#) method to remove a button. The ButtonClick event is fired if the user presses the drop down button of an editor. In this case, the Key parameter is empty. Use the [Change](#) event to notify your application that the editor's content is altered. If the editor hosts an ActiveX control use the [UserEditorOleEvent](#) event to monitor the events that inside ActiveX object fires.

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

C#	<pre>private void ButtonClick(object sender,exontrol.EXRECORDLib.Editor Ed,object Key) { }</pre>
VB	<pre>Private Sub ButtonClick(ByVal sender As System.Object,ByVal Ed As exontrol.EXRECORDLib.Editor,ByVal Key As Object) Handles ButtonClick End Sub</pre>

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

C#	<pre>private void ButtonClick(object sender, AxEXRECORDLib._IRecordEvents_ButtonClickEvent e) { }</pre>
C++	<pre>void OnButtonClick(LPDISPATCH Ed,VARIANT Key) { }</pre>
C++ Builder	<pre>void __fastcall ButtonClick(TObject *Sender,Exrecordlib_tlb::IEditor *Ed,Variant Key) { }</pre>



**Delphi** procedure ButtonClick(ASender: TObject; Ed : IEditor;Key : OleVariant);  
begin  
end;

**Delphi 8  
(.NET  
only)** procedure ButtonClick(sender: System.Object; e:  
AxEXRECORDLib.\_IRecordEvents\_ButtonClickEvent);  
begin  
end;

**Powe...** begin event ButtonClick(oleobject Ed,any Key)  
end event ButtonClick

**VB.NET** Private Sub ButtonClick(ByVal sender As System.Object, ByVal e As  
AxEXRECORDLib.\_IRecordEvents\_ButtonClickEvent) Handles ButtonClick  
End Sub

**VB6** Private Sub ButtonClick(ByVal Ed As EXRECORDLibCtl.IEditor,ByVal Key As Variant)  
End Sub

**VBA** Private Sub ButtonClick(ByVal Ed As Object,ByVal Key As Variant)  
End Sub

**VFP** LPARAMETERS Ed,Key

**Xbas...** PROCEDURE OnButtonClick(oRecord,Ed,Key)  
RETURN

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

**Java...** <SCRIPT EVENT="ButtonClick(Ed,Key)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function ButtonClick(Ed,Key)  
End Function  
</SCRIPT>

Visual  
Data...

```
Procedure OnComButtonClick Variant IIEd Variant IIKey  
    Forward Send OnComButtonClick IIEd IIKey  
End_Procedure
```

Visual  
Objects

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

X++

```
void onEvent_ButtonClick(COM _Ed,COMVariant _Key)  
{  
}
```

XBasic

```
function ButtonClick as v (Ed as OLE::Exontrol.Record.1::IEditor,Key as A)  
end function
```

dBASE

```
function nativeObject_ButtonClick(Ed,Key)  
return
```

The following VB sample displays the key of the button being clicked:

```
Private Sub Form_Load()  
    With Record1  
        With .Add("Calc", CalculatorType)  
            .Value = 3.14  
            .AddButton "AKey", , 1  
            .AddButton "BKey", , 2  
        End With  
    End With  
End Sub  
  
Private Sub Record1_ButtonClick(ByVal Ed As EXRECORDLibCtl.IEditor, ByVal Key As  
Variant)  
    Debug.Print Ed.Label & ", Key = """" & Key & """"  
End Sub
```

The following VC sample displays the key of the button being clicked:

```
ColeVariant vtMissing; vtMissing.vt = VT_ERROR;  
CEditor editor = m_record.Add(ColeVariant("Calc"), /*CalculatorType*/ 21, vtMissing );
```

```
editor.SetValue( COleVariant( 3.14 ) );  
editor.AddButton( COleVariant( "AKey" ), vtMissing, COleVariant( (long) 1 ), vtMissing,  
vtMissing, vtMissing );  
editor.AddButton( COleVariant( "BKey" ), vtMissing, COleVariant( (long) 1 ), vtMissing,  
vtMissing, vtMissing );
```

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )  
{  
    if ( pv )  
    {  
        if ( pv->vt == VT_ERROR )  
            return szDefault;  
  
        COleVariant vt;  
        vt.ChangeType( VT_BSTR, pv );  
        return V_BSTR( &vt );  
    }  
    return szDefault;  
}
```

```
void OnButtonClickRecord1(LPDISPATCH Ed, const VARIANT FAR& Key)  
{  
    CEditor editor( Ed );  
    editor.m_bAutoRelease = FALSE;  
  
    TCHAR szOutput[1024];  
    wsprintf( szOutput, "%s, Key = \"%s\\\"\\n", (LPCTSTR)editor.GetLabel(), (LPCTSTR)V2S( &  
(VARIANT&)Key ) );  
    OutputDebugString( szOutput );  
}
```

## event Change (Ed as Editor, NewValue as Variant)

Occurs when the user changes the editor's content.

Type	Description
Ed as <a href="#">Editor</a>	An Editor object whose value is changed.
NewValue as Variant	A Variant expression that indicates the newly editor's value.

The Change event notifies your application that the user changes the editor's value. Use the [ButtonClick](#) event to notify your application that the user clicks the editor's button. If the editor hosts an ActiveX control use the [UserEditorOleEvent](#) event to monitor the events inside the ActiveX object. If the control is bounded to a database using the [DataSource](#) property, the control automatically updates the database. If failed, the [LastError](#) property gets the description of the last error. The Change event is fired just before changing the [Value](#) property. The Value property specifies the value of the editor. The field's value depends on the type of the editor that's assigned to the field. For instance, if the field has assigned a DropDownListType editor, the Value property indicates a long expression that indicates the index of the predefined item being selected. Use the [Caption](#) property to retrieve the caption of the editor. Use the [Label](#) property to retrieve the label of the editor.

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

```
C# private void Change(object sender,exontrol.EXRECORDLib.Editor Ed,ref object  
    NewValue)  
{  
}
```

```
VB Private Sub Change(ByVal sender As System.Object,ByVal Ed As  
    exontrol.EXRECORDLib.Editor,ByRef NewValue As Object) Handles Change  
End Sub
```

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

```
C# private void Change(object sender, AxEXRECORDLib._IRecordEvents_ChangeEvent  
    e)  
{  
}
```

```
C++ void OnChange(LPDISPATCH Ed,VARIANT FAR* NewValue)  
{
```

```
}
```

C++  
Builder

```
void __fastcall Change(TObject *Sender,Exrecordlib_tlb::IEditor *Ed,Variant *  
NewValue)  
{  
}
```

Delphi

```
procedure Change(ASender: TObject; Ed : IEditor;var NewValue : OleVariant);  
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure Change(sender: System.Object; e:  
AxEXRECORDLib._IRecordEvents_ChangeEvent);  
begin  
end;
```

Powe...

```
begin event Change(oleobject Ed,any NewValue)  
end event Change
```

VB.NET

```
Private Sub Change(ByVal sender As System.Object, ByVal e As  
AxEXRECORDLib._IRecordEvents_ChangeEvent) Handles Change  
End Sub
```

VB6

```
Private Sub Change(ByVal Ed As EXRECORDLibCtl.IEditor,NewValue As Variant)  
End Sub
```

VBA

```
Private Sub Change(ByVal Ed As Object,NewValue As Variant)  
End Sub
```

VFP

```
LPARAMETERS Ed,NewValue
```

Xbas...

```
PROCEDURE OnChange(oRecord,Ed,NewValue)  
RETURN
```

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

Java...

```
<SCRIPT EVENT="Change(Ed,NewValue)" LANGUAGE="JScript">
```

```
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function Change(Ed,NewValue)  
End Function  
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComChange Variant IIEd Variant IINewValue  
    Forward Send OnComChange IIEd IINewValue  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_Change(Ed,NewValue) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Change(COM _Ed,COMVariant /*variant*/ _NewValue)  
{  
}
```

XBasic

```
function Change as v (Ed as OLE::Exontrol.Record.1::IEditor,NewValue as A)  
end function
```

dBASE

```
function nativeObject_Change(Ed,NewValue)  
return
```

The following VB sample displays the editor's value as soon as the user changes the control's content:

```
Private Sub Record1_Change(ByVal Ed As EXRECORDLibCtl.IEditor, NewValue As Variant)  
    Debug.Print Ed.Label & " = " & NewValue  
End Sub
```

The following VC sample displays the editor's value as soon as the user changes the control's content:

```
void OnChangeRecord1(LPDISPATCH Ed, VARIANT FAR* NewValue)  
{  
    CEditor editor( Ed );  
    editor.m_bAutoRelease = FALSE;
```

```
TCHAR szOutput[1024];  
wsprintf( szOutput, "%s, = %s", (LPCTSTR)editor.GetLabel(), (LPCTSTR)V2S( NewValue ) );  
OutputDebugString( szOutput );  
}
```

# 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 and [DbClick](#) 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. The [ButtonClick](#) event notifies your application that the user clicks a button. The [RClick](#) event notifies your application when user right clicks the control.

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;
```

Power...

```
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(oRecord)  
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 DbtClick (Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS)

Occurs when the user dbtclk the left mouse button over an object.

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 DbtClick event is fired when the user dbl clicks on the control. Use the DbtClick event to notify your application that a cell has been double-clicked. Use the [EditorFromPoint](#) property to get the editor from the point.

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

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

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

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

```
C# private void DbtClick(object sender,
AxEXRECORDLib._IRecordEvents_DbtClickEvent e)
{
}
```

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

```
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:
AxEXRECORDLib._IRecordEvents_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
AxEXRECORDLib._IRecordEvents_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(oRecord,Shift,X,Y)
RETURN
```

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

Java...

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

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

```
Visual  
Data... Procedure OnComDbClick Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS  
IYY  
Forward Send OnComDbClick IIShift IIX IYY  
End_Procedure
```

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

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

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

```
dBASE function nativeObject_DbClick(Shift,X,Y)  
return
```

The following VB sample displays the editor being double clicked:

```
Private Sub Record1_DbClick(Shift As Integer, X As Single, Y As Single)  
Dim e As EXRECORDLibCtl.Editor  
Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)  
If Not e Is Nothing Then  
Debug.Print e.Label  
End If  
End Sub
```

The following VC sample displays the editor being double clicked:

```
void OnDbClickRecord1(short Shift, long X, long Y)  
{
```

```
CEditor editor = m_record.GetEditorFromPoint( X, Y );  
if ( editor.m_lpDispatch != NULL )  
{  
    TCHAR szOutput[1024];  
    wsprintf( szOutput, "%s", (LPCTSTR)editor.GetLabel() );  
    OutputDebugString( szOutput );  
}  
}
```

# 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,
AxEXRECORDLib._IRecordEvents_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:  
AxEXRECORDLib._IRecordEvents_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  
AxEXRECORDLib._IRecordEvents_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(oRecord,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,
AxEXRECORDLib._IRecordEvents_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:  
AxEXRECORDLib.\_IRecordEvents\_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  
AxEXRECORDLib.\_IRecordEvents\_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(oRecord,KeyAscii)  
RETURN

Syntax for KeyPress event, **ICOM** 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 Integer KeyAscii  
    Forward Send OnComKeyPress Integer KeyAscii  
End_Procedure
```

Visual  
Objects

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

C++

```
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){}

VBPrivate Sub KeyUp(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal Shift As Short) Handles KeyUpEnd Sub

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

C#private void KeyUpEvent(object sender,AxEXRECORDLib.\_IRecordEvents\_KeyUpEvent e){}

C++void OnKeyUp(short FAR\* KeyCode,short Shift){}

C++ Buildervoid \_\_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:  
AxEXRECORDLib._IRecordEvents_KeyUpEvent);  
begin  
end;
```

Power...

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

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As  
AxEXRECORDLib._IRecordEvents_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(oRecord,KeyCode,Shift)  
RETURN
```

Syntax for KeyUp event, **ICOM** 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 Integer KeyCode Short Integer Shift
    Forward Send OnComKeyUp Integer KeyCode Integer Shift
End_Procedure
```

Visual  
Objects

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

C++

```
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 a 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 [DbClick](#) 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. Use the [EditorFromPoint](#) property to get the editor from point.

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,
AxEXRECORDLib._IRecordEvents_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: AxEXRECORDLib.\_IRecordEvents\_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 AxEXRECORDLib.\_IRecordEvents\_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(oRecord,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 IButton Short IShift OLE\_XPOS\_PIXELS IIX  
OLE\_YPOS\_PIXELS IY  
    Forward Send OnComMouseDown IButton IShift IIX IY  
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.Record.1::OLE\_XPOS\_PIXELS,Y as  
OLE::Exontrol.Record.1::OLE\_YPOS\_PIXELS)  
end function

**dBASE** function nativeObject\_MouseDown(Button,Shift,X,Y)  
return

The following VB sample displays the editor from the point when user clicks the control:

```
Private Sub Record1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim e As EXRECORDLibCtl.Editor
    Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not e Is Nothing Then
```

```
    MsgBox e.Label  
End If  
End Sub
```

The following VC sample displays the editor from the point when user clicks the control:

```
void OnMouseDownRecord1(short Button, short Shift, long X, long Y)  
{  
    CEditor editor = m_record.GetEditorFromPoint( X, Y );  
    if ( editor.m_lpDispatch != NULL )  
    {  
        TCHAR szOutput[1024];  
        wsprintf( szOutput, "%s", (LPCTSTR)editor.GetLabel() );  
        ::MessageBox( NULL, szOutput, NULL, NULL );  
    }  
}
```

# 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. Use the [EnsureVisible](#) method to ensures that an editor fits the control's client area. Use the [EditorFromPoint](#) property to get the editor from the cursor.

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

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

VBPrivate Sub MouseMoveEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseMoveEventEnd Sub

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

C#private void MouseMoveEvent(object sender,AxEXRECORDLib.\_IRecordEvents\_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: AxEXRECORDLib.\_IRecordEvents\_MouseMoveEvent);  
begin  
end;

**PowerBASIC** 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 AxEXRECORDLib.\_IRecordEvents\_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

**Xbasic** PROCEDURE OnMouseMove(oRecord,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 IButton Short IShift OLE\_XPOS\_PIXELS IIX  
OLE\_YPOS\_PIXELS IY  
    Forward Send OnComMouseMove IButton IShift IIX IY  
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.Record.1::OLE\_XPOS\_PIXELS,Y as  
OLE::Exontrol.Record.1::OLE\_YPOS\_PIXELS)  
end function

**dBASE** function nativeObject\_MouseMove(Button,Shift,X,Y)  
return

The following VB sample prints the editor from the point:

```
Private Sub Record1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim e As EXRECORDLibCtl.Editor
    Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not e Is Nothing Then
```

```
    Debug.Print e.Label & " = " & e.Value
End If
End Sub
```

The following VC sample prints the editor from the point:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

void OnMouseMoveRecord1(short Button, short Shift, long X, long Y)
{
    CEditor editor = m_record.GetEditorFromPoint( X, Y );
    if ( editor.m_lpDispatch != NULL )
    {
        TCHAR szOutput[1024];
        wsprintf( szOutput, "%s = %s\n", (LPCTSTR)editor.GetLabel(), (LPCTSTR)V2S(
&editor.GetValue() ) );
        OutputDebugString( szOutput );
    }
}
```

## event MouseUp (Button as Integer, Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS)

Occurs when the user releases a 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 [DbClick](#) 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. Use the [EditorFromPoint](#) method to retrieve the item from point.

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,
AxEXRECORDLib._IRecordEvents_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: AxEXRECORDLib.\_IRecordEvents\_MouseUpEvent);  
begin  
end;

**PowerBuilder** 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 AxEXRECORDLib.\_IRecordEvents\_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

**Xbase++** PROCEDURE OnMouseUp(oRecord,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 IButton Short IShift OLE_XPOS_PIXELS IIX  
OLE_YPOS_PIXELS IY  
    Forward Send OnComMouseUp IButton IShift IIX IY  
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.Record.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.Record.1::OLE_YPOS_PIXELS)  
end function`

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

The following VB sample displays a message when user right clicks the control:

```
Private Sub Record1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If (Button = vbRightButton) Then
        Dim e As EXRECORDLibCtl.Editor
        Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
        If Not e Is Nothing Then
```

```
        MsgBox e.Label
    End If
End If
End Sub
```

The following VC sample displays a message when user right clicks the control:

```
void OnMouseUpRecord1(short Button, short Shift, long X, long Y)
{
    if ( Button == 2 )
    {
        CEditor editor = m_record.GetEditorFromPoint( X, Y );
        if ( editor.m_lpDispatch != NULL )
        {
            TCHAR szOutput[1024];
            wsprintf( szOutput, "%s", (LPCTSTR)editor.GetLabel() );
            ::MessageBox( NULL, szOutput, NULL, NULL );
        }
    }
}
```

# event RClick ()

Fired when right mouse button is clicked.

Type	Description
------	-------------

The RClick event is fired each time the user releases the right mouse button over the control. Use the [MouseDown](#) event in case you need the position of the cursor when right clicking the control.

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

```
C# private void RClick(object sender)
{
}
```

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

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

```
C# private void RClick(object sender, EventArgs e)
{
}
```

```
C++ void OnRClick()
{
}
```

```
C++ Builder void __fastcall RClick(TObject *Sender)
{
}
```

```
Delphi procedure RClick(ASender: TObject; );
begin
end;
```

```
Delphi 8 (.NET only) procedure RClick(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe... begin event RClick()  
end event RClick

VB.NET Private Sub RClick(ByVal sender As System.Object, ByVal e As System.EventArgs)  
Handles RClick  
End Sub

VB6 Private Sub RClick()  
End Sub

VBA Private Sub RClick()  
End Sub

VFP LPARAMETERS nop

Xbas... PROCEDURE OnRClick(oRecord)  
RETURN

Syntax for RClick event, **ICOM** version (others), on:

Java... <SCRIPT EVENT="RClick()" LANGUAGE="JScript">  
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">  
Function RClick()  
End Function  
</SCRIPT>

Visual  
Data... Procedure OnComRClick  
Forward Send OnComRClick  
End\_Procedure

Visual  
Objects METHOD OCX\_RClick() CLASS MainDialog  
RETURN NIL

X++ void onEvent\_RClick()  
{

```
}
```

XBasic

```
function RClick as v ()  
end function
```

dBASE

```
function nativeObject_RClick()  
return
```

The following VB sample displays a message when user right clicks the control:

```
Private Sub Record1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)  
    If (Button = vbRightButton) Then  
        Dim e As EXRECORDLibCtl.Editor  
        Set e = Record1.EditorFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)  
        If Not e Is Nothing Then  
            MsgBox e.Label  
        End If  
    End If  
End Sub
```

The following VC sample displays a message when user right clicks the control:

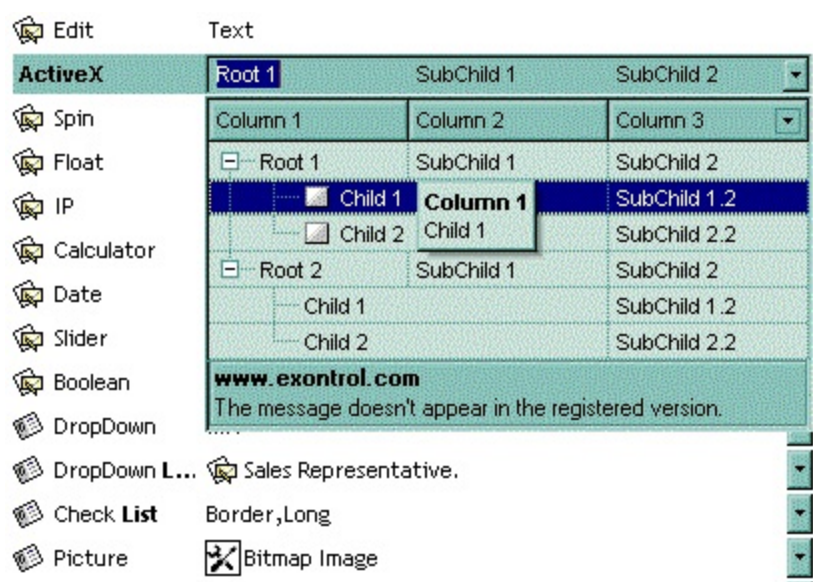
```
void OnMouseUpRecord1(short Button, short Shift, long X, long Y)  
{  
    if ( Button == 2 )  
    {  
        CEditor editor = m_record.GetEditorFromPoint( X, Y );  
        if ( editor.m_lpDispatch != NULL )  
        {  
            TCHAR szOutput[1024];  
            wsprintf( szOutput, "%s", (LPCTSTR)editor.GetLabel() );  
            ::MessageBox( NULL, szOutput, NULL, NULL );  
        }  
    }  
}
```

# event UserEditorOleEvent (Object as Object, Ev as OleEvent, Ed as Editor)

Occurs when an user editor fires an event.

Type	Description
Object as Object	An Object that fires the event.
Ev as <a href="#">OleEvent</a>	An OleEvent object that holds information about fired event.
Ed as <a href="#">Editor</a>	An Editor object whose inner ActiveX control fires the event.

The UserEditorOleEvent event notifies your application when an inner ActiveX control fires an event. The UserEditorType type specifies an editor that may host an ActiveX control. Use the [Add](#) method to insert an editor that hosts an ActiveX control. Use the [UserEditor](#) method to create an inner ActiveX control.



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

**C#**

```
private void UserEditorOleEvent(object sender,object
Obj,exontrol.EXRECORDLib.OleEvent Ev,exontrol.EXRECORDLib.Editor Ed)
{
}
```

**VB**

```
Private Sub UserEditorOleEvent(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Ev As exontrol.EXRECORDLib.OleEvent,ByVal Ed As
exontrol.EXRECORDLib.Editor) Handles UserEditorOleEvent
End Sub
```

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

C#	<pre>private void UserEditorOleEvent(object sender, AxEXRECORDLib._IRecordEvents_UserEditorOleEventEvent e) { }</pre>
C++	<pre>void OnUserEditorOleEvent(LPDISPATCH Object,LPDISPATCH Ev,LPDISPATCH Ed) { }</pre>
C++ Builder	<pre>void __fastcall UserEditorOleEvent(TObject *Sender,IDispatch *Object,Exrecordlib_tlb::IOleEvent *Ev,Exrecordlib_tlb::IEditor *Ed) { }</pre>
Delphi	<pre>procedure UserEditorOleEvent(ASender: TObject; Object : IDispatch;Ev : IOleEvent;Ed : IEditor); begin end;</pre>
Delphi 8 (.NET only)	<pre>procedure UserEditorOleEvent(sender: System.Object; e: AxEXRECORDLib._IRecordEvents_UserEditorOleEventEvent); begin end;</pre>
Powe...	<pre>begin event UserEditorOleEvent(oleobject Object,oleobject Ev,oleobject Ed) end event UserEditorOleEvent</pre>
VB.NET	<pre>Private Sub UserEditorOleEvent(ByVal sender As System.Object, ByVal e As AxEXRECORDLib._IRecordEvents_UserEditorOleEventEvent) Handles UserEditorOleEvent End Sub</pre>
VB6	<pre>Private Sub UserEditorOleEvent(ByVal Object As Object,ByVal Ev As EXRECORDLibCtl.IOleEvent,ByVal Ed As EXRECORDLibCtl.IEditor) End Sub</pre>



VBA

```
Private Sub UserEditorOleEvent(ByVal Object As Object,ByVal Ev As Object,ByVal Ed As Object)
End Sub
```

VFP

```
LPARAMETERS Object,Ev,Ed
```

Xbas...

```
PROCEDURE OnUserEditorOleEvent(oRecord,Object,Ev,Ed)
RETURN
```

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

Java...

```
<SCRIPT EVENT="UserEditorOleEvent(Object,Ev,Ed)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function UserEditorOleEvent(Object,Ev,Ed)
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComUserEditorOleEvent Variant IObject Variant IIEv Variant IIEd
    Forward Send OnComUserEditorOleEvent IObject IIEv IIEd
End_Procedure
```

Visual  
Objects

```
METHOD OCX_UserEditorOleEvent(Object,Ev,Ed) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_UserEditorOleEvent(COM _Object,COM _Ev,COM _Ed)
{
}
```

XBasic

```
function UserEditorOleEvent as v (Object as P,Ev as
OLE::Exontrol.Record.1::IOleEvent,Ed as OLE::Exontrol.Record.1::IEditor)
end function
```

dBASE

```
function nativeObject_UserEditorOleEvent(Object,Ev,Ed)
return
```

The following VB sample adds an [Exontrol.ComboBox](https://www.exontrol.com/excombobox.jsp) control and displays the events being fired by inner ActiveX control:

```
Option Explicit

Private Function isInstalled(ByVal s As String) As Boolean
On Error GoTo Error
    CreateObject (s)
    isInstalled = True
    Exit Function
Error:
    isInstalled = False
End Function

Private Sub Form_Load()
    With Record1
        .BeginUpdate
        With .Add("ActiveX", UserEditorType)
            .Position = 2
            Dim progID As String
            progID = "Exontrol.ComboBox"
            If Not (isInstalled(progID)) Then
                .Value = "" & progID & "" is not installed."
                .ToolTip = .Value
                .ForeColor = vbRed
            Else
                .UserEditor progID, ""
                .LabelBackColor = SystemColorConstants.vbMenuBar
                ' Accesses the inside ActiveX control, in our case an ExComboBox control.
                https://www.exontrol.com/excombobox.jsp
                With .UserEditorObject()
                    .BeginUpdate
                    .BackColorEdit = SystemColorConstants.vbMenuBar
                    .IntegralHeight = True
                    .ColumnAutoResize = True
                    .LinesAtRoot = True
```

```

.MinHeightList = 164
.MinWidthList = 264
.MarkSearchColumn = False
.FilterBarDropDownHeight = -150
.DrawGridLines = True
.Alignment = 0
With .Columns
    .Add "Column 1"
    .Add "Column 2"
    With .Add("Column 3")
        .DisplayFilterButton = True
    End With
End With
With .Items
    Dim h, h1
    h = .AddItem(Array("Root 1", "SubChild 1", "SubChild 2"))
    h1 = .InsertItem(h, , Array("Child 1", "SubChild 1.1", "SubChild 1.2"))
    .CellMerge(h1, 0) = 1
    .CellHasCheckBox(h1, 0) = True
    h1 = .InsertItem(h, , Array("Child 2", "SubChild 2.1", "SubChild 2.2"))
    .CellMerge(h1, 0) = 1
    .CellHasCheckBox(h1, 0) = True
    .ExpandItem(h) = True
    h = .AddItem(Array("Root 2", "SubChild 1", "SubChild 2"))
    h1 = .InsertItem(h, , Array("Child 1", "SubChild 1.1", "SubChild 1.2"))
    .CellMerge(h1, 0) = 1
    h1 = .InsertItem(h, , Array("Child 2", "SubChild 2.1", "SubChild 2.2"))
    .CellMerge(h1, 0) = 1
    .ExpandItem(h) = True
End With
.Value = "Root 1"
.EndUpdate
End With
End If
End With
.EndUpdate
End With

```

```
End Sub
```

```
Private Sub Record1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As  
EXRECORDLibCtl.IOleEvent, ByVal Ed As EXRECORDLibCtl.IEditor)
```

```
On Error Resume Next
```

```
    Debug.Print "Event name: " & Ev.Name
```

```
    If (Ev.CountParam = 0) Then
```

```
        Debug.Print vbTab & "The event has no arguments."
```

```
    Else
```

```
        Debug.Print "The event has the following arguments:"
```

```
        Dim i As Long
```

```
        For i = 0 To Ev.CountParam - 1
```

```
            Debug.Print vbTab & Ev(i).Name; " = " & Ev(i).Value
```

```
        Next
```

```
    End If
```

```
End Sub
```

The following VC sample adds an [Exontrol.ComboBox](#) control and displays the events being fired by inner ActiveX control:

```
#import "c:\winnt\system32\ExComboBox.dll"
```

```
#import "c:\winnt\system32\ExRecord.dll"
```

```
CString strObject( "Exontrol.ComboBox" );
```

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
```

```
m_record.BeginUpdate();
```

```
m_record.SetLabelSize( 110 );
```

```
CEditor editor = m_record.Add( COleVariant( "ActiveX" ), EXRECORDLib::UserEditorType,  
vtMissing );
```

```
editor.SetPosition( 2 );
```

```
if ( !isInstalled( strObject.AllocSysString() ) )
```

```
{
```

```
    CString strFormat;
```

```
    strFormat.Format( "\"%s\" is not installed.", (LPCSTR)strObject );
```

```
    editor.SetValue( COleVariant( strFormat ) );
```

```
    editor.SetForeColor( RGB( 255, 0, 0 ) );
```

```
}
```

```
else
```

```
{
```

```
// Creates the exComboBox control. https://www.exontrol.com/excombobox.jsp
editor.UserEditor( strObject, "" );
if ( EXCOMBOBOXLib::IComboBoxPtr spComboBox = editor.GetUserEditorObject() )
{
    spComboBox->BeginUpdate();
    spComboBox->BackColorEdit = GetSysColor( COLOR_MENU );
    spComboBox->IntegralHeight = true;
    spComboBox->ColumnAutoResize = true;
    spComboBox->LinesAtRoot = EXCOMBOBOXLib::exLinesAtRoot;
    spComboBox->MinHeightList = 164;
    spComboBox->MinWidthList = 264;
    spComboBox->MarkSearchColumn = false;
    spComboBox->DrawGridLines = EXCOMBOBOXLib::exAllLines;
    spComboBox->FilterBarDropDownHeight = -150;
    spComboBox->Alignment = EXCOMBOBOXLib::RightAlignment;
    EXCOMBOBOXLib::IColumnsPtr spColumns = spComboBox->Columns;
    spColumns->Add("Column 1");
    spColumns->Add("Column 2");
    EXCOMBOBOXLib::IColumnPtr spColumn = spColumns->Add("Column 3");
    spColumn->DisplayFilterButton = true;
    EXCOMBOBOXLib::IItemsPtr spltems = spComboBox->Items;
    long h = spltems->AddItem( v( "Root 1" ) );
    spltems->CellCaption[v(h)][v((long)1)] = v("SubChild 1");
    spltems->CellCaption[v(h)][v((long)2)] = v("SubChild 2");
    long h1 = spltems->InsertItem( h, vtMissing, v( "Child 1" ) );
    spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 1.1");
    spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 1.2");
    spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
    spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
    h1 = spltems->InsertItem( h, vtMissing, v( "Child 2" ) );
    spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 2.1");
    spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 2.2");
    spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
    spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
    spltems->put_ExpandItem( h, TRUE );
}
```

```

h = spltems->AddItem( v( "Root 2" ) );
spltems->CellCaption[v(h)][v((long)1)] = v("SubChild 1");
spltems->CellCaption[v(h)][v((long)2)] = v("SubChild 2");
h1 = spltems->InsertItem( h, vtMissing, v( "Child 1" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 1.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 1.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
h1 = spltems->InsertItem( h, vtMissing, v( "Child 2" ) );
spltems->CellCaption[v(h1)][v((long)1)] = v("SubChild 2.1");
spltems->CellCaption[v(h1)][v((long)2)] = v("SubChild 2.2");
spltems->CellHasCheckBox[v(h1)][v((long)0)] = true;
spltems->CellMerge[v(h1)][v((long)0)] = v((long)1);
spltems->put_ExpandItem( h, TRUE );

```

```

spComboBox->Value = "Root 1";
spComboBox->EndUpdate();

```

```

}
}
m_record.EndUpdate();

```

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

```

void OnUserEditorOleEventRecord1(LPDISPATCH Object, LPDISPATCH Ev, LPDISPATCH Ed)

```

```

{
    EXRECORDLib::IOleEventPtr spEvent = Ev;
    CString strOutput = "Event name: ";
    strOutput += spEvent->Name;
    strOutput += "\r\n";
    if ( spEvent->CountParam == 0 )
    {
        strOutput += "\tThe event has no arguments.";
    }
    else
    {
        strOutput += "\tThe event has no arguments.\r\n";
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            strOutput += spEvent->GetParam( v(i) )->Name;
            strOutput += " = ";
            strOutput += V2S( &spEvent->GetParam( v(i) )->Value);
            strOutput += "\r\n";
        }
    }
    OutputDebugString( strOutput );
}

```

In C++, the **#import "path-to-ExRecord.dll"** adds a new EXRECORDLib namespace that includes definition for [OleEvent](#) and [OleEventParam](#) classes.