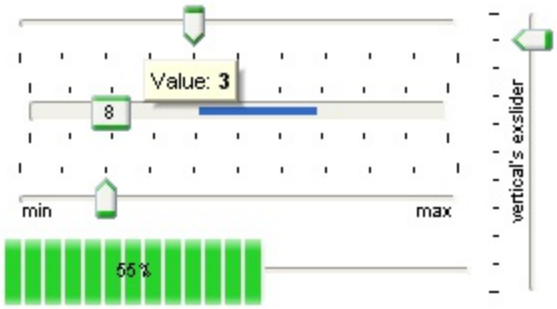
ExSlider

The Exontrol's eXSlider component allows you adding skinable sliders to your forms or dialogs. A "slider control" (also known as a trackbar) is a window containing a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the control sends events to indicate the change. The volume controls in the Windows operating system are slider controls. The eXSlider 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 control is written from scratch as most of our controls, and it is not sub classing a slider or trackbar window class or similar, that's why it supports features never seen in other controls.

Features include:

- WYSWYG Template/Layout Editor support
- Skinnable Interface support (ability to apply a skin to the any background part)
- Windows XP Theme support
- Horizontal or Vertical orientation support
- Customizable HTML labels
- Allow Floating Points
- ProgressBar support
- Owner Draw support
- Multi-lines tooltip support
- Ability to put HTML text on any part of the control, includes icons or custom size pictures
- ANSI and UNICODE versions available
- and more



How to get support?

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

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

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

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

Regards,
Exontrol Development Team

https://www.exontrol.com

constants AlignmentEnum

The AlignmentEnum type defines the caption's alignment. Use the Caption property to specify a text being displayed on any part of the control. Use the CaptionAlignment property to specify the alignment of the text inside the part.

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

constants AppearanceEnum

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

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

constants BackgroundPartEnum

The BackgroundPartEnum type defines the parts of the control in a specified state. Use the Background property to change the visual appearance of a any part of the control in a specified state.

Use the VisiblePart or VisibleParts property to specify which part is visible and which part is not visible. Use the EnablePart or EnableParts property to specify which part is enabled and which part is disabled.

- All BackgroundPartEnum expressions that starts with **exVS** changes a part in a vertical slider (Orientation property is exVertical).
- All BackgroundPartEnum expressions that starts with **exHS** changes a part in the horizontal slider (Orientation property is exHorizontal).
- Any BackgroundPartEnum expression that ends with **P** specifies a part of the control when it is pressed.
- Any BackgroundPartEnum expression that ends with **D** specifies a part of the control when it is disabled.
- Any BackgroundPartEnum expression that ends with **H** specifies a part of the control when the cursor hovers it.
- Any BackgroundPartEnum expression that ends with no **H**, **P** or **D** specifies a part of the control in normal state.

Name	Value	Description
exToolTipAppearance	64	Indicates the visual appearance of the borders of the tooltips. Use the ToolTipWidth property to specify the width of the tooltip window.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.
exVSSel	280	The selection range part (exSelPart) in normal state.
exVSSelP	281	The selection range part (exSelPart) when it is pressed.
exVSSelD	282	The selection range part (exSelPart) when it is disabled.

exVSSelH	283	The selection range part (exSelPart) when cursor hovers it.
exVSThumb	260	Specifies the thumb part (exThumbPart) in normal state.
exVSThumbP	261	Specifies the thumb part (exThumbPart) when it is pressed.
exVSThumbD	262	Specifies the thumb part (exThumbPart) when it is disabled.
exVSThumbH	263	Specifies the thumb part (exThumbPart) when cursor hovers it.
exVSLower	268	Specifies the lower part (exLowerBackPart) in normal state.
exVSLowerP	269	Specifies the lower part (exLowerBackPart) when it is pressed.
exVSLowerD	270	Specifies the lower part (exLowerBackPart) when it is disabled.
exVSLowerH	271	Specifies the lower part (exLowerBackPart) when the cursor hovers it.
exVSUpper	272	Specifies the upper part (exUpperBackPart) in normal state.
exVSUpperP	273	Specifies the upper part (exUpperBackPart) when it is pressed.
exVSUpperD	274	Specifies the upper part (exUpperBackPart) when it is disabled.
exVSUpperH	275	Specifies the upper part (exUpperBackPart) when the cursor hovers it.
exVSBack	276	Specifies the background part (exLowerBackPart and exUpperBackPart) in normal state.
exVSBackP	277	Specifies the background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exVSBackD	278	Specifies the background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exVSBackH	279	Specifies the background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exHSSel	408	The select range part (exSelPart) in normal state.
		The selection range part (exSelPart) when it is

exHSSelP	409	pressed.
exHSSelD	410	The selection range part (exSelPart) when it is disabled.
exHSSelH	411	The selection range part (exSelPart) when the cursor hovers it.
exHSThumb	388	Specifies the thumb part (exThumbPart) in normal state.
exHSThumbP	389	Specifies the thumb part (exThumbPart) when it is pressed.
exHSThumbD	390	Specifies the thumb part (exThumbPart) when it is disabled.
exHSThumbH	391	Specifies the thumb part (exThumbPart) when the cursor hovers it.
exHSLower	396	Specifies the lower part (exLowerBackPart) in normal state.
exHSLowerP	397	Specifies the lower part (exLowerBackPart) when it is pressed.
exHSLowerD	398	Specifies the lower part (exLowerBackPart) when it is disabled.
exHSLowerH	399	Specifies the lower part (exLowerBackPart) when the cursor hovers it.
exHSUpper	400	Specifies the upper part (exUpperBackPart) in normal state.
exHSUpperP	401	Specifies the upper part (exUpperBackPart) when it is pressed.
exHSUpperD	402	Specifies the upper part (exUpperBackPart) when it is disabled.
exHSUpperH	403	Specifies the upper part (exUpperBackPart) when the cursor hovers it.
exHSBack	404	Specifies the background part (exLowerBackPart and exUpperBackPart) in normal state.
exHSBackP	405	Specifies the background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exHSBackD	406	Specifies the background part (exLowerBackPart and exUpperBackPart) when it is disabled.

exHSBackH	407	Specifies the background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.

constants OrientationEnum

Specifies how the control is displayed. Use the Orientation property to specify the control's orientation.

Name	Value	Description
exVertical	0	The control is vertically oriented.
exHorizontal	1	The control is horizontally oriented.
exReverseOrder	16	exReverseOrder. The control displays the values in reverse order.

constants PartEnum

The PartEnum expression indicates the parts in the control. Use the Background property to change the visual aspect for a specified part. Use the VisiblePart or VisibleParts property to specify which part is visible and which part is not visible. Use the EnablePart or EnableParts property to specify which part is enabled and which part is disabled.

Name	Value	Description
exLowerBackPart	512	The area from the start to the thumb.
exThumbPart	256	The thumb/slider part
exUpperBackPart	128	The area between thumb and the end.
exBackgroundPart	640	The lower and upper parts.
exSelPart	1	The selection range part. Use the SelectRange property to make it visible.
exPartNone	0	exPartNone

constants PictureDisplayEnum

Specifies how the picture is displayed on the control's background. Use the PictureDisplay property to specify how the control displays its picture.

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	Tiles the picture on the source.

constants TickStyleEnum

Specifies how the control displays the ticks. Use the TickStyle property to specify how the ticks are displayed. The TickColor property indicates the color to paint the ticks.

Name	Value	Description
exBottomRight	0	The ticks are displayed on the bottom/right side.
exTopLeft	1	The ticks are displayed on the top/left side.
exBoth	2	The ticks are displayed on the both side.
exNoTicks	3	No ticks are displayed.

Appearance object

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

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

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

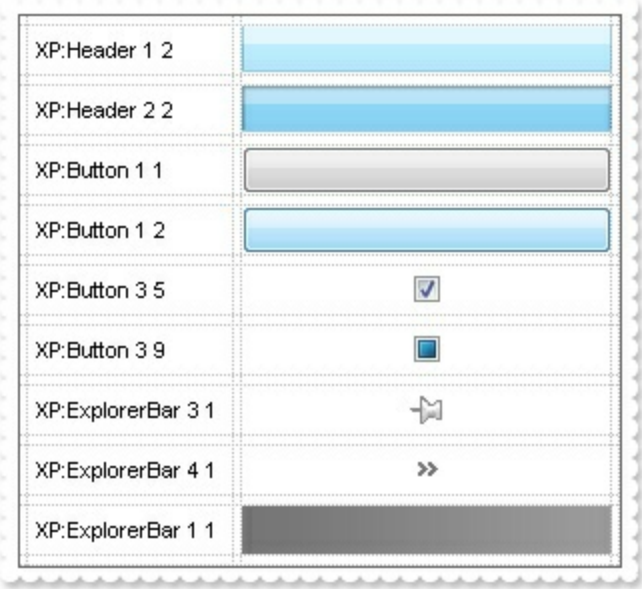
Adds or replaces a skin object to the control.

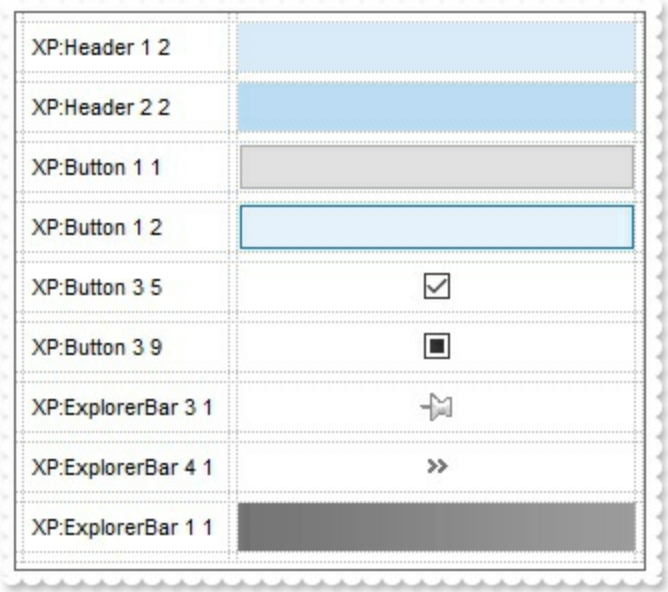
Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
	The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use the BYTE[] / safe arrays of VT_I1 or VT_UI1 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)) If the Skin parameter points to a string expression, it can be one of the following: • A path to the skin file (*.EBN). The ExButton component or ExEBN tool can be used to create, view or edit EBN files. For instance, "C:\Program Files\Exontrol\ExButton\Sample\EBN\MSOffice-Ribbon\msor_frameh.ebn" • A BASE64 encoded string that holds the skin file (*.EBN). Use the ExImages tool to build BASE 64 encoded strings of the skin file (*.EBN). The BASE64 encoded string starts with "gBFLBCJw..." • An Windows XP theme part, if the Skin parameter starts with "**XP:**". Use this option, to display any UI element of the Current Windows XP Theme, on any part of the control. In this case, the syntax of the Skin parameter is: "XP: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 of the part to be shown. All known values for window/class, part and start are defined at

the end of this document. For instance the "XP:Header 1 2" indicates the part 1 of the Header class in the state 2, in the current Windows XP theme.

The following screen shots show a few Windows XP Theme Elements, running on Windows Vista and Windows 10, using the XP options:

Skin as Variant

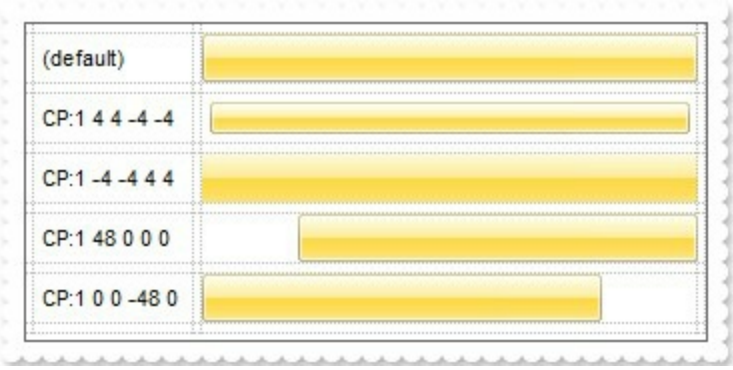




- A copy of another skin with different coordinates (position, size), if the Skin parameter starts with "**CP**:". Use this option, to display the EBN, using different coordinates (position, size). By default, the EBN skin object is rendered on the part's client area. Using this option, you can display the same EBN, on a different position / size. In this case, the syntax of the Skin parameter is: "CP:ID Left Top Right Bottom"

where the ID is the identifier of the EBN to be used (it is a number that specifies the ID parameter of the Add method), Left, Top, Right and Bottom parameters/numbers specifies the relative position to the part's client area, where the EBN should be rendered. The Left, Top, Right and Bottom parameters are numbers (negative, zero or positive values, with no decimal), that can be followed by the D character which indicates the value according to the current DPI settings. For instance, "CP:1 -2 -2 2 2", uses the EBN with the identifier 1, and displays it on a 2-pixels wider rectangle no matter of the DPI settings, while "CP:1 -2D -2D 2D 2D" displays it on a 2-pixels wider rectangle if DPI settings is 100%, and on on a 3-pixels wider rectangle if DPI settings is 150%.

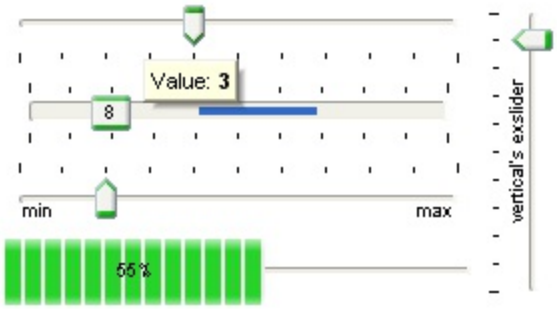
The following screen shot shows the same EBN being displayed, using different CP options:



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. Use the Background property to assign a skin or a color to any part of the control in a specified state.

For instance, the Background(exVSThumbP) = RGB(255,0,0) defines the thumb in a red color, when it is pressed. 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.



The identifier you choose for the skin is very important to be used in the background properties like explained bellow. Shortly, the color properties (Background property) 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 Background(exThumbPart) = Background(exThumbPart) Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the thumb part.

In the following samples, we have used the following skin file:  , and we get a slider like follows: 

The following VB sample changes the visual appearance of the thumb, in the vertical slider:

```
With Slider1
   .VisualAppearance.Add 1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn"
   .Background(exVSThumb) = &H1000000
End With
```

The following VB sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
With Slider1
   .VisualAppearance.Add 1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn"
   .Background(exVSThumbP) = &H1000000
End With
```

The following C++ sample changes the visual appearance of the thumb, in the vertical slider:

```
m_slider.GetVisualAppearance().Add( 1, COleVariant(
_T("D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn") ) );
m_slider.SetBackground( 260 /*exVSThumb*/, 0x01000000 );
```

The following C++ sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
m_slider.GetVisualAppearance().Add( 1, COleVariant(
_T("D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn") ) );
m_slider.SetBackground( 261 /*exVSThumbP*/, 0x01000000 );
```

The following VB.NET sample changes the visual appearance of the thumb, in the vertical slider:

```
With AxSlider1
   .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumb, &H1000000)
End With
```

The following VB.NET sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
With AxSlider1
   .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumbP, &H1000000)
End With
```

The following C# sample changes the visual appearance of the thumb, in the vertical slider:

```
axSlider1.VisualAppearance.Add(1, "D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical
2\\thumb.ebn");
axSlider1.set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumb, 0x1000000);
```

The following C# sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
axSlider1.VisualAppearance.Add(1, "D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical
2\\thumb.ebn");
axSlider1.set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumbP, 0x1000000);
```

The following VFP sample changes the visual appearance of the thumb, in the vertical slider:

```
with thisform.Slider1
   .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .Background(260) = 0x1000000
endwith
```

The following VFP sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
with thisform.Slider1
   .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .Background(261) = 0x1000000
endwith
```

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. The skin method may change the visual appearance for any part of the control, in any state.

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. The skin method may change the visual appearance for any part of the control, in any state.

Slider 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: {031F9B36-1219-4DF5-8E09-1A50B8185BC2}. The object's program identifier is: "Exontrol.Slider". The /COM object module is: "ExSlider.dll"

The Exontrol's eXSlider component allows you adding skinable sliders to your forms or dialogs. A "slider control" (also known as a trackbar) is a window containing a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the control sends events to indicate the change. The volume controls in the Windows operating system are slider controls. The component supports the following properties and methods:

Name	Description
AllowFloat	Specifies whether the slider's range includes floating numbers.
Appearance	Retrieves or sets the control's appearance.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
BackColor	Specifies the control's background color.
Background	Returns or sets a value that indicates the background color for parts in the control.
BeginUpdate	This method prevents the control from painting until the EndUpdate method is called.
Caption	Specifies the caption of the part of the control.
CaptionAlignment	Specifies the alignment of the part's caption.
CaptionIndentX	Indents the caption on x axis.
CaptionIndentY	Indents the caption on y axis.
Enabled	Enables or disables the control.
EnablePart	Indicates whether the specified part is enabled or disabled.
EnableParts	Specifies the parts of the control to be enabled or disabled.
EndUpdate	Resumes painting the control after painting is suspended by the BeginUpdate method.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.

Font	Retrieves or sets the control's font.
ForeColor	Specifies the control's foreground color.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
Images	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
ImageSize	Retrieves or sets the size of icons the control displays..
LabelTick	Specifies the label to be shown on ticks.
LargeChange	Gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance.
LargeChangeF	Gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance(as float).
Maximum	The upper limit value of the scrollable range.
MaximumF	The upper limit value of the scrollable range(as float).
Minimum	The lower limit value of the scrollable range.
MinimumF	The lower limit value of the scrollable range(as float).
NotifyParent	Specifies whether the control sends notifications to the parent window.
Orientation	Specifies the control's orientation.
OwnerDrawPart	Indicates which part of the control is responsible for its drawing.
PartFromPoint	Retrieves the part from the point.
Picture	Retrieves or sets a graphic to be displayed in the control.
PictureDisplay	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background
ReplaceIcon	Adds a new icon, replaces an icon or clears the control's image list.
SelectRange	Returns or sets a value that indicates whether the control can have a select range.
SelLength	Returns or sets the length of a selection.
SelStart	Returns or sets a value that indicates where a selection starts.

ShowFocusRect	Sets or gets a value that indicates whether the control is marked when it gets the focus.
ShowImageList	Specifies whether the control's image list window is visible or hidden.
ShowThumbProgress	Specifies whether the thumb indicates a progress bar.
ShowToolTip	Shows the specified tooltip at given position.
SmallChange	Gets or sets the value added to or subtracted from the Value property when the slider is moved a small distance.
SmallChangeF	Gets or sets the value added to or subtracted from the Value property when the slider is moved a small distance(as float).
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.
ThumbSize	Specifies the width or the height of the thumb.
TickColor	Specifies the color for the control's ticks.
TickFrequency	Returns or sets a value that indicates the ratio of ticks on the control.
TickFrequencyF	Returns or sets a value that indicates the ratio of ticks on the control.
TickStyle	Specifies where the ticks appears on the control.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipText	Specifies the control's tooltip text.
ToolTipTitle	Specifies the title of the control's tooltip.
ToolTipWidth	Specifies a value that indicates the width of the tooltip window, in pixels.
ToolTipX	Indicates an expression that determines the horizontal-position of the tooltip, in screen coordinates.
ToolTipY	Indicates an expression that determines the vertical-position of the tooltip, in screen coordinates.
UserData	Associates an extra data to a part of the control.
Value	The value that the thumb box position represents.
	The value that the thumb box position represents (as float

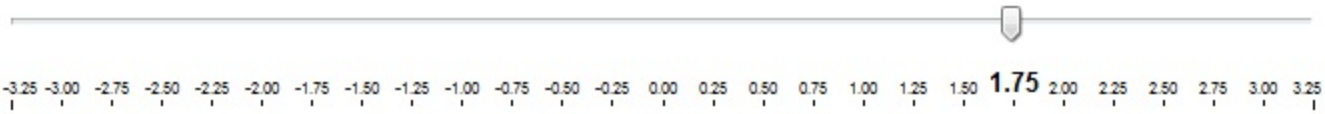
ValueF	)(as float).
ValueFromPoint	Retrieves the value from the point.
ValueFromPointF	Retrieves the value from the point (as float).
Version	Retrieves the control's version.
VisiblePart	Indicates whether the specified part is visible or hidden.
VisibleParts	Specifies the parts of the control being visible.
VisualAppearance	Retrieves the control's appearance.

property Slider.AllowFloat as Boolean

Specifies whether the slider's range includes floating numbers.

Type	Description
Boolean	A Boolean expression that indicates whether the slider displays floating numbers.

By default, the AllowFloat property is False. Use the AllowFloat property to use the control on handling value within a floating numbers range.



If the AllowFloat property is False (by default) the following properties work with integer numbers:

- Minimum property specifies the lower limit value of the scrollable range.
- Maximum property specifies the upper limit value of the scrollable range.
- Value property indicates the position of the slider.
- SmallChange property gets or sets the value added to or subtracted from the Value property when the slider is moved a small distance.
- LargeChange property indicates the amount by which the slider position changes when the user clicks in the slider or presses the PAGE UP or PAGE DOWN keys.
- TickFrequency property returns or sets a value that indicates the ratio of ticks on the control.

If the AllowFloat property is **True** (by default) the following properties work with floating numbers:

- MinimumF property specifies the lower limit value of the scrollable range.
- MaximumF property specifies the upper limit value of the scrollable range.
- ValueF property indicates the position of the slider.
- SmallChangeF property gets or sets the value added to or subtracted from the Value property when the slider is moved a small distance.
- LargeChangeF property indicates the amount by which the slider position changes when the user clicks in the slider or presses the PAGE UP or PAGE DOWN keys.
- TickFrequencyF property returns or sets a value that indicates the ratio of ticks on the control.

The following samples shows how to use and specify floating numbers within the control (the range from -3.25 to 3.25):

VBA (MS Access, Excell...)

```
' Change event - Occurs when the value of the control is changed.
Private Sub Slider1_Change()
   With Slider1
      Debug.Print( .ValueF )
   End With
End Sub

With Slider1
   .BeginUpdate
   .AllowFloat = True
   .MinimumF = -3.25
   .MaximumF = 3.25
   .SmallChangeF = 0.25
   .LargeChangeF = 1
   .TickFrequencyF = 0.5
   .ValueF = 0
   .LabelTick = "value format '' "
   .EndUpdate
End With
```

VB6

```
' Change event - Occurs when the value of the control is changed.
Private Sub Slider1_Change()
   With Slider1
      Debug.Print( .ValueF )
   End With
End Sub

With Slider1
   .BeginUpdate
   .AllowFloat = True
   .MinimumF = -3.25
   .MaximumF = 3.25
   .SmallChangeF = 0.25
   .LargeChangeF = 1
```

```
    .TickFrequencyF = 0.5
    .ValueF = 0
    .LabelTick = "value format '' "
    .EndUpdate
End With
```

VB.NET

```
' Change event - Occurs when the value of the control is changed.
Private Sub Exslider1_Change(ByVal sender As System.Object) Handles
Exslider1.Change
    With Exslider1
        Debug.Print( .ValueF )
    End With
End Sub

With Exslider1
    .BeginUpdate()
    .AllowFloat = True
    .MinimumF = -3.25
    .MaximumF = 3.25
    .SmallChangeF = 0.25
    .LargeChangeF = 1
    .TickFrequencyF = 0.5
    .ValueF = 0
    .LabelTick = "value format '' "
    .EndUpdate()
End With
```

VB.NET for /COM

```
' Change event - Occurs when the value of the control is changed.
Private Sub AxSlider1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxSlider1.Change
    With AxSlider1
        Debug.Print( .ValueF )
    End With
End Sub
```

```
With AxSlider1
    .BeginUpdate()
    .AllowFloat = True
    .MinimumF = -3.25
    .MaximumF = 3.25
    .SmallChangeF = 0.25
    .LargeChangeF = 1
    .TickFrequencyF = 0.5
    .ValueF = 0
    .LabelTick = "value format '' "
    .EndUpdate()
End With
```

C++

```
// Change event - Occurs when the value of the control is changed.
void OnChangeSlider1()
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXSLIDERLib' for the library: 'ExSlider 1.0 Control
Library'
        #import <ExSlider.dll>
        using namespace EXSLIDERLib;
    */
    EXSLIDERLib::ISliderPtr spSlider1 = GetDlgItem(IDC_SLIDER1)-
>GetControlUnknown();
    OutputDebugStringW( _bstr_t(spSlider1->GetValueF()) );
}

EXSLIDERLib::ISliderPtr spSlider1 = GetDlgItem(IDC_SLIDER1)-
>GetControlUnknown();
spSlider1->BeginUpdate();
spSlider1->PutAllowFloat(VARIANT_TRUE);
spSlider1->PutMinimumF(-3.25);
spSlider1->PutMaximumF(3.25);
```

```
spSlider1->PutSmallChangeF(0.25);
spSlider1->PutLargeChangeF(1);
spSlider1->PutTickFrequencyF(0.5);
spSlider1->PutValueF(0);
spSlider1->PutLabelTick(L"value format '' ");
spSlider1->EndUpdate();
```

C++ Builder

```
// Change event - Occurs when the value of the control is changed.
void __fastcall TForm1::Slider1Change(TObject *Sender)
{
   OutputDebugString( PChar(Slider1->ValueF) );
}

Slider1->BeginUpdate();
Slider1->AllowFloat = true;
Slider1->MinimumF = -3.25;
Slider1->MaximumF = 3.25;
Slider1->SmallChangeF = 0.25;
Slider1->LargeChangeF = 1;
Slider1->TickFrequencyF = 0.5;
Slider1->ValueF = 0;
Slider1->LabelTick = L"value format '' ";
Slider1->EndUpdate();
```

C#

```
// Change event - Occurs when the value of the control is changed.
private void exslider1_Change(object sender)
{
   System.Diagnostics.Debug.Print( exslider1.ValueF.ToString() );
}
//this.exslider1.Change += new
exontrol.EXSLIDERLib.exg2antt.ChangeEventHandler(this.exslider1_Change);
```

```
exslider1.BeginUpdate();
exslider1.AllowFloat = true;
exslider1.MinimumF = -3.25;
exslider1.MaximumF = 3.25;
exslider1.SmallChangeF = 0.25;
exslider1.LargeChangeF = 1;
exslider1.TickFrequencyF = 0.5;
exslider1.ValueF = 0;
exslider1.LabelTick = "value format '' ";
exslider1.EndUpdate();
```

JavaScript

```
<SCRIPT FOR="Slider1" EVENT="Change()" LANGUAGE="JScript">
   alert( Slider1.ValueF );
</SCRIPT>

<OBJECT classid="clsid:031F9B36-1219-4DF5-8E09-1A50B8185BC2" id="Slider1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   Slider1.BeginUpdate();
   Slider1.AllowFloat = true;
   Slider1.MinimumF = -3.25;
   Slider1.MaximumF = 3.25;
   Slider1.SmallChangeF = 0.25;
   Slider1.LargeChangeF = 1;
   Slider1.TickFrequencyF = 0.5;
   Slider1.ValueF = 0;
   Slider1.LabelTick = "value format '' ";
   Slider1.EndUpdate();
</SCRIPT>
```

C# for /COM

```
// Change event - Occurs when the value of the control is changed.
private void axSlider1_Change(object sender, EventArgs e)
```

```
{
    System.Diagnostics.Debug.Print( axSlider1.ValueF.ToString() );
}
//this.axSlider1.Change += new EventHandler(this.axSlider1_Change);

axSlider1.BeginUpdate();
axSlider1.AllowFloat = true;
axSlider1.MinimumF = -3.25;
axSlider1.MaximumF = 3.25;
axSlider1.SmallChangeF = 0.25;
axSlider1.LargeChangeF = 1;
axSlider1.TickFrequencyF = 0.5;
axSlider1.ValueF = 0;
axSlider1.LabelTick = "value format '' ";
axSlider1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
// Change event - Occurs when the value of the control is changed.
void onEvent_Change()
{
    ;
    print( exslider1.ValueF() );
}

public void init()
{
    ;

    super();

    exslider1.BeginUpdate();
    exslider1.AllowFloat(true);
    exslider1.MinimumF(-3.25);
    exslider1.MaximumF(3.25);
    exslider1.SmallChangeF(0.25);
```

```
    exslider1.LargeChangeF(1);
    exslider1.TickFrequencyF(0.5);
    exslider1.ValueF(0);
    exslider1.LabelTick("value format '' ");
    exslider1.EndUpdate();
}
```

Delphi 8 (.NET only)

```
// Change event - Occurs when the value of the control is changed.
procedure TWinForm1.AxSlider1_Change(sender: System.Object; e:
System.EventArgs);
begin
   with AxSlider1 do
   begin
      OutputDebugString( ValueF );
   end
end;

with AxSlider1 do
begin
   BeginUpdate();
   AllowFloat := True;
   MinimumF := -3.25;
   MaximumF := 3.25;
   SmallChangeF := 0.25;
   LargeChangeF := 1;
   TickFrequencyF := 0.5;
   ValueF := 0;
   LabelTick := 'value format '''' ';
   EndUpdate();
end
```

Delphi (standard)

```
// Change event - Occurs when the value of the control is changed.
procedure TForm1.Slider1Change(ASender: TObject; );
begin
```

```
  with Slider1 do
  begin
    OutputDebugString( ValueF );
  end
end;

with Slider1 do
begin
  BeginUpdate();
  AllowFloat := True;
  MinimumF := -3.25;
  MaximumF := 3.25;
  SmallChangeF := 0.25;
  LargeChangeF := 1;
  TickFrequencyF := 0.5;
  ValueF := 0;
  LabelTick := 'value format '''' ';
  EndUpdate();
end
```

VFP

```
*** Change event - Occurs when the value of the control is changed. ***
LPARAMETERS nop
  with thisform.Slider1
    DEBUGOUT( .ValueF )
  endwith

with thisform.Slider1
  .BeginUpdate
  .AllowFloat = .T.
  .MinimumF = -3.25
  .MaximumF = 3.25
  .SmallChangeF = 0.25
  .LargeChangeF = 1
  .TickFrequencyF = 0.5
  .ValueF = 0
```

```
   .LabelTick = "value format '' "
   .EndUpdate
endwith
```

dBASE Plus

```
/*
with (this.ACTIVEX1.nativeObject)
   Change = class::nativeObject_Change
endwith
*/
// Occurs when the value of the control is changed.
function nativeObject_Change()
   local oSlider
   oSlider = form.Activex1.nativeObject
   ? Str(oSlider.ValueF)
return

local oSlider

oSlider = form.Activex1.nativeObject
oSlider.BeginUpdate()
oSlider.AllowFloat = true
oSlider.MinimumF = -3.25
oSlider.MaximumF = 3.25
oSlider.SmallChangeF = 0.25
oSlider.LargeChangeF = 1
oSlider.TickFrequencyF = 0.5
oSlider.ValueF = 0
oSlider.LabelTick = "value format '' "
oSlider.EndUpdate()
```

Visual Objects

```
METHOD OCX_Exontrol1Change() CLASS MainDialog
   // Change event - Occurs when the value of the control is changed.
   OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:ValueF) ))
```

```
RETURN NIL


oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:AllowFloat := true
oDCOCX_Exontrol1:MinimumF := -3.25
oDCOCX_Exontrol1:MaximumF := 3.25
oDCOCX_Exontrol1:SmallChangeF := 0.25
oDCOCX_Exontrol1:LargeChangeF := 1
oDCOCX_Exontrol1:TickFrequencyF := 0.5
oDCOCX_Exontrol1:ValueF := 0
oDCOCX_Exontrol1:LabelTick := "value format '' "
oDCOCX_Exontrol1:EndUpdate()
```

PowerBuilder

```
/*begin event Change() - Occurs when the value of the control is changed.*/
/*
   OleObject oSlider
   oSlider = ole_1.Object
   MessageBox("Information",string( String(oSlider.ValueF) ))
*/
/*end event Change*/

OleObject oSlider

oSlider = ole_1.Object
oSlider.BeginUpdate()
oSlider.AllowFloat = true
oSlider.MinimumF = -3.25
oSlider.MaximumF = 3.25
oSlider.SmallChangeF = 0.25
oSlider.LargeChangeF = 1
oSlider.TickFrequencyF = 0.5
oSlider.ValueF = 0
oSlider.LabelTick = "value format '' "
```

```
oSlider.EndUpdate()
```

property Slider.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

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

By default, the control displays no border. Use the VisualAppearance property to add new skins to the control. Use the Background property to change the visual appearance for a specific part of the control.

The following VB sample changes the visual aspect of the borders of the control (please check the above picture for round corners):

```
With Slider1
   .BeginUpdate
      .VisualAppearance.Add &H16, "c:\temp\frame.ebn"
      .Appearance = &H16000000
      .BackColor = RGB(250, 250, 250)
   .EndUpdate
End With
```

The following VB.NET sample changes the visual aspect of the borders of the control:

```
With AxSlider1
   .BeginUpdate()
```

```
    .VisualAppearance.Add(&H16, "c:\temp\frame.ebn")
    .Appearance = &H16000000
    .BackColor = Color.FromArgb(250, 250, 250)
    .EndUpdate()
End With
```

The following C# sample changes the visual aspect of the borders of the control:

```
axSlider1.BeginUpdate();
axSlider1.VisualAppearance.Add(0x16, "c:\\temp\\frame.ebn");
axSlider1.Appearance = (EXSLIDERLib.AppearanceEnum)0x16000000;
axSlider1.BackColor = Color.FromArgb(250, 250, 250);
axSlider1.EndUpdate();
```

The following C++ sample changes the visual aspect of the borders of the control:

```
m_slider.BeginUpdate();
m_slider.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\frame.ebn" ) );
m_slider.SetAppearance( 0x16000000 );
m_slider.SetBackColor( RGB(250,250,250) );
m_slider.EndUpdate();
```

The following VFP sample changes the visual aspect of the borders of the control:

```
with thisform.Slider1
    .BeginUpdate
        .VisualAppearance.Add(0x16, "c:\temp\frame.ebn")
        .Appearance = 0x16000000
        .BackColor = RGB(250, 250, 250)
    .EndUpdate
endwith
```

method Slider.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 Slider1_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 Slider.BackColor as Color

Specifies the control's background color.

Type	Description
Color	A Color expression that indicates the

Use the BackColor property to specify the control's background color. This property does not affect the visual appearance of the control applied using the Background property. Use the Picture property to assign a picture on the control's background. Use the ForeColor property to specify the control's foreground color. The Caption property assigns a text on any part of the control.

property Slider.Background(Part as BackgroundPartEnum) as Color

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

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

The Background property specifies a background color or a visual appearance for specific parts in the control. If the Background property is 0, the control draws the part as default. Use the Add method to add new skins to the control. Use the Remove method to remove a specific skin from the control. Use the Clear method to remove all skins in the control. Use the BeginUpdate and EndUpdate methods to maintain performance while multiple changes are applied. Use the VisiblePart or VisibleParts property to specify visible parts in the control.

In the following samples, we have used the following skin file:  and we get the following slider: 

The following VB sample changes the visual appearance of the thumb, in the vertical slider:

```
With Slider1
   .VisualAppearance.Add 1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn"
   .Background(exVSThumb) = &H1000000
End With
```

The following VB sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
With Slider1
   .VisualAppearance.Add 1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn"
```

```
    .Background(exVSThumbP) = &H1000000
End With
```

The following C++ sample changes the visual appearance of the thumb, in the vertical slider:

```
m_slider.GetVisualAppearance().Add( 1, COleVariant(
_T("D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn") ) );
m_slider.SetBackground( 260 /*exVSThumb*/, 0x01000000 );
```

The following C++ sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
m_slider.GetVisualAppearance().Add( 1, COleVariant(
_T("D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn") ) );
m_slider.SetBackground( 261 /*exVSThumbP*/, 0x01000000 );
```

The following VB.NET sample changes the visual appearance of the thumb, in the vertical slider:

```
With AxSlider1
    .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn")
    .set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumb, &H1000000)
End With
```

The following VB.NET sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
With AxSlider1
    .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical
2\thumb.ebn")
    .set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumbP, &H1000000)
End With
```

The following C# sample changes the visual appearance of the thumb, in the vertical slider:

```
axSlider1.VisualAppearance.Add(1, "D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical
2\\thumb.ebn");
axSlider1.set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumb, 0x1000000);
```

The following C# sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
axSlider1.VisualAppearance.Add(1, "D:\\Exontrol\\ExSlider\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn");
axSlider1.set_Background(EXSLIDERLib.BackgroundPartEnum.exVSThumbP, 0x1000000);
```

The following VFP sample changes the visual appearance of the thumb, in the vertical slider:

```
with thisform.Slider1
   .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical 2\thumb.ebn")
   .Background(260) = 0x1000000
endwith
```

The following VFP sample changes the visual appearance of the thumb (**when it is pressed**), in the vertical slider:

```
with thisform.Slider1
   .VisualAppearance.Add(1, "D:\Exontrol\ExSlider\sample\VB\Gauge\Vertical 2\thumb.ebn")
   .Background(261) = 0x1000000
endwith
```

method Slider.BeginUpdate ()

This method prevents the control from painting until the EndUpdate method is called.

Type	Description

This method prevents the control from painting until the EndUpdate method is called. The BeginUpdate and EndUpdate methods increases the speed of making your changes, by preventing painting the control when it suffers any change. Once that BeginUpdate method was called, you have to make sure that EndUpdate method will be called too.

property Slider.Caption(Part as PartEnum) as String

Specifies the caption of the part of the control.

Type	Description
Part as PartEnum	A PartEnum expression that specifies the part where the text is displayed.
String	A String expression that indicates the text being displayed. The Caption property support built-in HTML format as explained bellow.

Use the Caption property to specify a caption on any part of the control. Use the Font property to specify the control's font. Use the ForeColor property to specify the caption's color, if the <fgcolor> tag is not used. Use the Value property to specify the control's value. The CaptionAlignment property specifies the alignment of the caption in the part area. Use the CaptionIndentX property to indent the caption on the part, on the X axis. Use the CaptionIndentY property to indent the caption of the part on the Y axis. Use the Background property to change the visual appearance for any part of the control, in any state. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

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

- ** ... ** displays the text in **bold**
- ** ... ** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** ~~Strike~~-through text
- **<a id;options> ... ** 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.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<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
- **
** 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 **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € 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 bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a

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

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

For instance, the following VB sample prints the control's Value on the control's thumb:

```
Private Sub Slider1_Change()
   With Slider1
      .Caption(exThumbPart) = .Value
   End With
End Sub
```

The following C++ sample prints the control's Value on the control's thumb:

```
void OnChangeSlider1()
```

```
{
    CString strFormat;
    strFormat.Format( _T("%i"), m_slider.GetValue() );
    m_slider.SetCaption( 256, strFormat );
}
```

The following VB.NET sample prints the control's Value on the control's thumb:

```
With AxSlider1
    .set_Caption(EXSLIDERLib.PartEnum.exThumbPart, .Value.ToString())
End With
```

The following C# sample prints the control's Value on the control's thumb:

```
private void axSlider1_Change(object sender, EventArgs e)
{
    axSlider1.set_Caption(EXSLIDERLib.PartEnum.exThumbPart, axSlider1.Value.ToString());
}
```

The following VFP sample prints the control's Value on the control's thumb:

```
*** ActiveX Control Event ***

with thisform.Slider1
    .Caption(256) = .Value
endwith
```

property Slider.CaptionAlignment(Part as PartEnum) as AlignmentEnum

Specifies the alignment of the part's caption.

Type	Description
Part as PartEnum	A PartEnum expression that specifies the part where the text is displayed.
AlignmentEnum	An AlignmentEnum expression that specifies the alignment of the caption.

By default, the CaptionAlignment property is CenterAlignment. Use the CaptionIndentX property to indent the caption on the part, on the X axis. Use the CaptionIndentY property to indent the caption of the part on the Y axis. Use the Caption property to specify a caption on any part of the control. Use the Font property to specify the control's font. Use the ForeColor property to specify the caption's color, if the <fgcolor> tag is not used. Use the Value property to specify the control's value.

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

- ** ... ** displays the text in **bold**
- ** ... ** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** ~~Strike~~-through text
- **<a id;options> ... ** 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.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<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
- **
** 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 **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € 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 bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient

color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

property Slider.CaptionIndentX(Part as PartEnum) as Long

Indents the caption on x axis.

Type	Description
Part as PartEnum	A PartEnum expression that specifies the part where the text is displayed.
Long	A long expression that specifies the indentation of the caption.

By default, the CaptionIndentX property is 0. Use the CaptionIndentX property to indent the caption on the part, on the X axis. Use the CaptionIndentY property to indent the caption of the part on the Y axis. Use the CaptionAlignment property to align the caption of the part. Use the Caption property to specify a caption on any part of the control. Use the Font property to specify the control's font. Use the ForeColor property to specify the caption's color, if the <fgcolor> tag is not used. Use the Value property to specify the control's value.

property Slider.CaptionIndentY(Part as PartEnum) as Long

Indents the caption on y axis.

Type	Description
Part as PartEnum	A PartEnum expression that specifies the part where the text is displayed.
Long	A long expression that specifies the indentation of the caption.

By default, the CaptionIndentX property is 0. Use the CaptionIndentY property to indent the caption of the part on the Y axis. Use the CaptionIndentX property to indent the caption on the part, on the X axis. Use the CaptionAlignment property to align the caption of the part. Use the Caption property to specify a caption on any part of the control. Use the Font property to specify the control's font. Use the ForeColor property to specify the caption's color, if the <fgcolor> tag is not used. Use the Value property to specify the control's value.

property Slider.Enabled as Boolean

Enables or disables the control.

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

By default, the Enabled property is True. Use the EnablePart or EnableParts property to specify a disabled part. If the Enabled property is False, all visible parts of the control are displayed in disabled state. Use the VisiblePart property to specify which parts are visible or hidden. Use the Background property to apply a visual effect on any part of the control in any state.

property Slider.EnablePart(Part as PartEnum) as Boolean

Indicates whether the specified part is enabled or disabled.

Type	Description
Part as PartEnum	A PartEnum expression that specifies the part being enabled or disabled.
Boolean	A Boolean expression that specifies whether the part is enabled or diasable.

By default, when a part becomes visible, automatically the EnablePart is called. Use the EnablePart property to disable parts of the control. A disabled part can't be clicked, and shows the disabled state. Use the Background property to apply a visual effect on any part of the control. The EnableParts property is similar with the EnablePart property. Use the VisiblePart property to specify which parts are visible or hidden. The ClickPart or ClickingPart event is fired only if the user clicked in an enabled part.

By default, the following parts are enabled:

- exLowerBackPart (the part between the start and the thumb part of the control)
- exThumbPart (the thumb/slider part)
- exUpperBackPart (the part between the thumb and end of the control)

property Slider.EnableParts as Long

Specifies the parts of the control to be enabled or disabled.

Type	Description
Long	A long expression that specifies an OR combination of PartEnum values that indicates which parts are visible and which parts are not shown.

By default, the EnableParts property is 897 (that's a OR combination of exLowerBackPart, exThumbPart, exUpperBackPart). The VisiblePart property specifies which part is visible and which part is hidden. By default, when a part becomes visible, automatically the EnablePart is called. Use the EnablePart property to disable parts of the control. A disabled part can't be clicked, and shows the disabled state. Use the Background property to apply a visual effect on any part of the control. The EnableParts property is similar with the EnablePart property.

By default, the following parts are enabled:

- exLowerBackPart (the part between the start and the thumb part of the control)
- exThumbPart (the thumb/slider part)
- exUpperBackPart (the part between the thumb and the end part of the control)

method Slider.EndUpdate ()

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

Type	Description

This method prevents the control from painting until the EndUpdate method is called. The BeginUpdate and EndUpdate methods increases the speed of making your changes, by preventing painting the control when it suffers any change. Once that BeginUpdate method was called, you have to make sure that EndUpdate method will be called too.

property Slider.EventParam(Parameter as Long) as Variant

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

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

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

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

In most languages you will type something like:

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

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

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

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

reference.

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

method Slider.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 control's background color:

```
Debug.Print Slider1.ExecuteTemplate("BackColor")
```

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 Slider.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object used to paint captions.

Use the Font property to specify the font being used when a part displays its caption. Use the Caption property to specify a text in any part of the control. Use the ForeColor property to specify the caption's color, if the <fgcolor> tag is not used. Use the Value property to specify the control's value. The CaptionAlignment property specifies the alignment of the caption in the part ar

property Slider.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. The Caption property assigns a text on any part of the control. Use the BackColor property to specify the control's background color. This property does not affect the visual appearance of the control applied using the Background property. Use the Picture property to assign a picture on the control's background.

property Slider.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

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

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

property Slider.hWnd as Long

Retrieves the control's window handle.

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

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

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

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

Use the Images method to add icons being displayed in any part of the control using the Caption property. The user can add images at design time, by drag and drop files to combo's image holder. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ReplaceIcon method to add, remove or clear icons in the control's images collection. Use the Caption property to specify the part's caption. Use the HTMLPicture property to display custom size pictures in any part of the control.

property Slider.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 Slider.LabelTick as String

Specifies the label to be shown on ticks.

Type	Description
String	A String expression that indicates the label to be displayed on ticks. The result value of the formatting expression supports built-in HTML format like explained bellow. For instance, use the LabelTick property on **"value"** to show the labels on each tick.

By default, the LabelTick property is empty, which means no labels are displayed on ticks. The TickFrequency/TickFrequencyF property returns or sets a value that indicates the ratio of ticks on the control. The AllowFloat property specifies whether the control integer or floating numbers.

For instance:

- "value", shows the values for each tick.
- " (value=current ? '<font ;12><fgcolor=FF0000>' : '') + value", shows the current slider's position with a different color and font.
- "value = current ? value : "", shows the value for the current tick only.
- "(value = current ? '<b><font ;10>' : '') + (value array 'ab bc cd de ef fg gh hi ij jk kl' split ' ')" displays different captions for slider's values.

The LabelTick property is a formatted expression which result may include the HTML tags.

The LabelTick property indicates a formatting expression that may use the following predefined keywords:

- **value** gets the slider's position to be displayed
- **current** gets the current slider's value. The current keyword gets the Value/ValueF
- **vmin** gets the slider's minimum value. The vmin keyword gets the Minimum/MinimumF
- **vmax** gets the slider's maximum value. The vmax keyword gets the Maximum/MaximumF
- **smin** gets the slider's selection minimum value. The smin keyword gets the SelStart value.
- **smax** gets the slider's selection maximum value. The smax keyword gets the SelStart + SelLength value.

The supported binary arithmetic operators are:

- ***** (multiplicity operator), priority 5
- ***/*** (divide operator), priority 5
- **mod** (reminder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

- **not** (not operator), priority 3 (high priority)

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (or operator), priority 1

The supported binary boolean operators, all these with the same priority 0, are :

- **<** (less operator)
- **<=** (less or equal operator)
- **=** (equal operator)
- **!=** (not equal operator)
- **>=** (greater or equal operator)
- **>** (greater operator)

The supported ternary operators, all these with the same priority 0, are :

- **?** (**Immediate If operator**), returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for is

  "expression ? true_part : false_part"

  , while it executes and returns the true_part if the expression is true, else it executes and returns the false_part. For instance, the *"%0 = 1 ? 'One' : (%0 = 2 ? 'Two' : 'not found')"* returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the case() statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- ***array*** *(at operator),* returns the element from an array giving its index (0 base). The *array* operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for *array* operator is

"expression array (c1,c2,c3,...cn)"

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *"month(value)-1 array ('J','F','M','A','M','Jun','J','A','S','O','N','D')"* is equivalent with *"month(value)-1 case (default:''; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D')"*.

- ***in*** *(include operator),* specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

"expression in (c1,c2,c3,...cn)"

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *"value in (11,22,33,44,13)"* is equivalent with *"(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)"*. The *in* operator is not a time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the *in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- ***switch*** *(switch operator),* returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

"expression switch (default,c1,c2,c3,...,cn)"

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the "%0 switch ('not found',1,4,7,9,11)" gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker that iif (immediate if operator) alterative.

- ***case()*** *(case operator)* returns and executes one of n expressions, depending on the evaluation of the expression (IIF - immediate IF operator is a binary case() operator). The syntax for *case()* operator is:

"expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)"

If the default part is missing, the case() operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, ... the default_expression is executed and returned. If the value of the expression is c1, then the *case()* operator executes and returns the *expression1.* The *default, c1, c2, c3, ...* must be constant elements as numbers, dates or strings. For instance, the "*date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)*" indicates that only *#1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002#* dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: "*date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)*" statement indicates the working hours for dates as follows:

-
  - #4/1/2009#, from hours 06:00 AM to 12:00 PM
  - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
  - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using iif and or expressions.

Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. For instance type(%0) = 8 specifies the cells that contains string values.

  Here's few predefined types:

  - 0 - empty (not initialized)
  - 1 - null
  - 2 - short
  - 3 - long
  - 4 - float
  - 5 - double
  - 6 - currency
  - 7 - date
  - 8 - string
  - 9 - object
  - 10 - error
  - 11 - boolean

    - 12 - variant
    - 13 - any
    - 14 - decimal
    - 16 - char
    - 17 - byte
    - 18 - unsigned short
    - 19 - unsigned long
    - 20 - long on 64 bits
    - 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS.

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2
- value **format** 'flags' (binary operator) formats the value with specified flags. If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the 1000 format '' displays 1,000.00 for English format, while 1.000,00 is displayed for German format. 1000 format '2|.|3|,' will always displays 1,000.00 no matter of settings in the control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

  The ' flags' for format operator is a list of values separated by | character such as '*NumDigits*|*DecimalSep*|*Grouping*|*ThousandSep*|*NegativeOrder*|*LeadingZero*' with the following meanings:

    - *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
    - *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
    - Grouping - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit

indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
  - *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
  - *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
    - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
    - 1 - Negative sign, number; for example, -1.1
    - 2 - Negative sign, space, number; for example, - 1.1
    - 3 - Number, negative sign; for example, 1.1-
    - 4 - Number, space, negative sign; for example, 1.1 -
  - *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string
- **lower** (unary operator) returns a string expression in lowercase letters
- **upper** (unary operator) returns a string expression in uppercase letters
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names
- **ltrim** (unary operator) removes spaces on the left side of a string
- **rtrim** (unary operator) removes spaces on the right side of a string
- **trim** (unary operator) removes spaces on both sides of a string
- **startwith** (binary operator) specifies whether a string starts with specified string
- **endwith** (binary operator) specifies whether a string ends with specified string
- **contains** (binary operator) specifies whether a string contains another specified string
- **left** (binary operator) retrieves the left part of the string
- **right** (binary operator) retrieves the right part of the string
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on)
- a **count** b (binary operator) retrieves the number of occurrences of the b in a
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result.
- a **split** b, splits the a using the separator b, and returns an array. For instance, the "weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' **split** ' '" gets the weekday as

string. This operator can be used with the array

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance the shortdateF(December 31, 1971 11:00 AM) returns "12/31/1971".
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format.
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel.
- **year** (unary operator) retrieves the year of the date (100,...,9999)
- **month** (unary operator) retrieves the month of the date (1, 2,...,12)
- **day** (unary operator) retrieves the day of the date (1, 2,...,31)
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365)
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday)
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23)
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59)
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59)

The LabelTick property can display labels using the following built-in HTML tags:

- **** displays the text in **bold**.
- **** displays the text in *italics*.
- **<u></u>** <u>underlines</u> the text.
- **<s></s>** ~~Strike~~-through text
- **** displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.
- **<fgcolor=RRGGBB></fgcolor>** displays text with a specified foreground color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bgcolor=RRGGBB></bgcolor>** displays text with a specified background color. The RR, GG or BB should be hexa values and indicates red, green and blue values.

- **
** a forced line-break
- **<solidline>** The next line shows a solid-line on top/bottom side. If has no effect for a single line caption.
- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single line caption.
- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text
- **<c>** Centers the text
- **<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 **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number** (the character with specified code), For instance, the &#8364 displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display bold in HTML caption you can use bold

property Slider.LargeChange as Long

The amount by which the slider position changes when the user clicks in the slider or presses the PAGE UP or PAGE DOWN keys.

Type	Description
Long	A long expression that indicates the value being added or subtracted from the control's Value when the user clicks the slider's upper or lower area.

By default, the LargeChange property is 10. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance. If the LargeChange property is 0, the Value property is not changed when clicking the the upper or lower part of the control. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The LabelTick property indicates the HTML expression to be displayed as labels for each tick.

The Value property goes from:

- Minimum to Maximum values

property Slider.LargeChangeF as Double

Gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance(as float).

Type	Description
Double	A floating expression that indicates the value being added or subtracted from the control's Value when the user clicks the slider's upper or lower area.

By default, the LargeChangeF property is 10.00 This property has effect ONLY if the AllowFloat property is True. The LargeChangeF property gets or sets a value to be added to or subtracted from the ValueF property when the slider is moved a large distance. The SmallChangeF property gets or sets the value added to or subtracted from the ValueF property when the thumb is moved a small distance. If the LargeChangeF property is 0, the ValueF property is not changed when clicking the the upper or lower part of the control. Use the MinimumF and MaximumF properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The LabelTick property indicates the HTML expression to be displayed as labels for each tick.

The ValueF property goes from:

- MinimumF to MaximumF values

property Slider.Maximum as Long

The upper limit value of the scrollable range.

Type	Description
Long	A long expression that indicates the upper limit value of the scrollable range.

By default, the Maximum property is 10. The Value property specifies the control's value. The Minimum property specifies the lower limit value of the scrollable range. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance.

The Value property goes from:

- Minimum to Maximum values

property Slider.MaximumF as Double

The upper limit value of the scrollable range(as float).

Type	Description
Double	A floating expression that indicates the upper limit value of the scrollable range.

By default, the MaximumF property is 10.00 The MaximumF property has effect ONLY if the AllowFloat property is True. The ValueF property specifies the control's value. The MinimumF property specifies the lower limit value of the scrollable range. The LargeChangeF property gets or sets a value to be added to or subtracted from the ValueF property when the slider is moved a large distance. The SmallChangeF property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance.

The ValueF property goes from:

- MinimumF to MaximumF values

property Slider.Minimum as Long

The lower limit value of the scrollable range.

Type	Description
Long	A long expression that indicates the lower limit value of the scrollable range.

By default, the Minimum property is 0. The Value property specifies the control's value. The Maximum property specifies the upper limit value of the scrollable range. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance.

The Value property goes from:

- Minimum to Maximum values

property Slider.MinimumF as Double

The lower limit value of the scrollable range(as float).

Type	Description
Double	A floating expression that indicates the lower limit value of the scrollable range.

By default, the MinimumF property is 0.00 This property has effect ONLY if the AllowFloat property is True. The ValueF property specifies the control's value. The MaximumF property specifies the upper limit value of the scrollable range. The LargeChangeF property gets or sets a value to be added to or subtracted from the ValueF property when the slider is moved a large distance. The SmallChangeF property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance.

The ValueF property goes from:

- MinimumF to MaximumF values

property Slider.NotifyParent as Boolean

Specifies whether the control sends notifications to the parent window.

Type	Description
Boolean	A Boolen expression that indicates whether the control sends notification messages to the parent window, when an event occurs.

Currently, this property is not implemented.

property Slider.Orientation as OrientationEnum

Specifies how the control displays the slider and the ticks.

Type	Description
OrientationEnum	An OrientationEnum expression that indicates the control's orientation.

By default, the Orientation property is exHorizontal. Use the Orientation property to change the control's orientation. Use the Value property to specify the control's value.

property Slider.OwnerDrawPart(Part as PartEnum) as Boolean

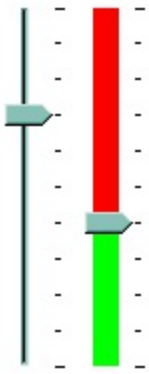
Indicates which part of the control is responsible for its drawing.

Type	Description
Part as PartEnum	A PartEnum expression that's responsible for its drawing
Boolean	A Boolean expression that indicates whether the user is responsible for drawing the specified part, or not.

By default, the OwnerDrawPart property is 0. The control fires the OwnerDrawStart and OwnerDrawEnd events when the control requires drawing the owner draw part. These events are fired only for visible parts, that have the OwnerDrawPart property on True. The VisiblePart or VisibleParts property specifies the part being visible or hidden.

The control paints the parts in the following order (only if visible):

- exBackgroundPart
- exLowerBackPart
- exUpperBackPart
- exSelPart
- exThumbPart



For instance, the following VB sample draws the lower part in red, and the upper part in green (as in the screen shot) :

```
With Slider1
   .OwnerDrawPart(exLowerBackPart Or exUpperBackPart) = True
End With
```

```
Private Type RECT
      Left As Long
      Top As Long
      Right As Long
```

```
    Bottom As Long
End Type
Private Declare Function GetClipBox Lib "gdi32" (ByVal hdc As Long, lpRect As RECT) As Long
Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long
Private Declare Function CreateSolidBrush Lib "gdi32" (ByVal crColor As Long) As Long
Private Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long

Private Sub Slider1_OwnerDrawEnd(ByVal Part As EXSLIDERLibCtl.PartEnum, ByVal hdc As Long)
  Dim r As RECT, h As Long
  GetClipBox hdc, r
  r.Left = r.Left + 4
  r.Right = r.Right - 4
  If Part = exLowerBackPart Then
    h = CreateSolidBrush(RGB(255, 0, 0))
    FillRect hdc, r, h
    DeleteObject (h)
  Else
    If Part = exUpperBackPart Then
      h = CreateSolidBrush(RGB(0, 255, 0))
      FillRect hdc, r, h
      DeleteObject (h)
    End If
  End If
End Sub
```

The following C++ sample draws the lower part in red, and the upper part in green (as in the screen shot) :

```
m_slider.SetOwnerDrawPart( 128 /*exUpperBackPart*/,  TRUE );
m_slider.SetOwnerDrawPart( 512 /*exLowerBackPart*/,  TRUE );
```

```
void OnOwnerDrawEndSlider1(long Part, long hDC)
{
  HDC h = (HDC)hDC;
  RECT rtPart = {0}; GetClipBox( h, &rtPart );
```

```
    InflateRect( &rtPart, -4, 0 );
    switch ( Part )
    {
        case 128: /*exUpperBackPart*/
        {
            HBRUSH hB = CreateSolidBrush( RGB(0,255,0) );
            FillRect( h, &rtPart, hB );
            DeleteObject( hB );
            break;
        }
        case 512: /*exLowerBackPart*/
        {
            HBRUSH hB = CreateSolidBrush( RGB(255,0,0) );
            FillRect( h, &rtPart, hB );
            DeleteObject( hB );
            break;
        }
    }
}
```

property Slider.PartFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as PartEnum

Retrieves the part from the point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
PartEnum	A PartEnum expression that indicates the part from the point.

The PartFromPoint property specifies the part of the control from the cursor. Use the ValueFromPoint property to determine the value from the cursor. Use the VisiblePart or VisibleParts property to specify the visible parts of the control.

The following VB sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With Slider1
      If (0 <> (.PartFromPoint(-1, -1) And exBackgroundPart)) Then
         .Value = .ValueFromPoint(-1, -1)
      End If
   End With
End Sub
```

The following VB.NET sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
Private Sub AxSlider1_MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_MouseDownEvent) Handles AxSlider1.MouseDownEvent
   With AxSlider1
      If (0 <> (.get_PartFromPoint(-1, -1) And EXSLIDERLib.PartEnum.exBackgroundPart))
Then
         .Value = .get_ValueFromPoint(-1, -1)
```

```
        End If
    End With
End Sub
```

The following C++ sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
void OnMouseDownSlider1(short Button, short Shift, long X, long Y)
{
    if ( m_slider.GetPartFromPoint(-1,-1) & 640 )
        m_slider.SetValue( m_slider.GetValueFromPoint(-1,-1) );
}
```

The following C# sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
private void axSlider1_MouseDownEvent(object sender,
AxEXSLIDERLib._ISliderEvents_MouseDownEvent e)
{
    if ( 0 != ( axSlider1.get_PartFromPoint(-1,-1) &
EXSLIDERLib.PartEnum.exBackgroundPart ) )
        axSlider1.Value = axSlider1.get_ValueFromPoint(-1, -1);
}
```

The following VFP sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform.slider1
    if ( 0 # bitand( .PartFromPoint(-1,-1), 640) )
        .Value = .ValueFromPoint(-1,-1)
    endif
endwith
```

property Slider.Picture as IPictureDisp

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

Type	Description
IPictureDisp	A Picture object that's displayed on the control's background.

By default, the control has no picture associated. The control uses the PictureDisplay property to determine how the picture is displayed on the control's background. Use the BackColor property to change the control's background color. Use the Background property to change the visual appearance for any part of the control in any state.

property Slider.PictureDisplay as PictureDisplayEnum

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

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the picture is displayed.

By default, the PictureDisplay property is exTile. Use the PictureDisplay property specifies how the Picture is displayed on the control's background. If the control has no picture associated the PictureDisplay property has no effect. Use the BackColor property to change the control's background color. Use the Background property to change the visual appearance for any part of the control in any state.

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

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

Type	Description
Icon as Variant	A long expression that indicates the icon's handle.
Index as Variant	A long expression that indicates the index where icon is inserted.

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

Use the ReplaceIcon property to add, remove or replace an icon in the control's images collection. Also, the ReplaceIcon property can clear the images collection. Use the Images method to attach a image list to the control. Use the Caption property to specify the part's caption. Use the HTMLPicture property to display custom size pictures in any part of the control.

The following VB sample adds a new icon to control's images list:

i = ExSlider1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle), i specifies the index where the icon is added

The following VB sample replaces an icon into control's images list::

i = ExSlider1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle, 0), i is zero, so the first icon is replaced.

The following VB sample removes an icon from control's images list:

ExSlider1.ReplaceIcon 0, i, i specifies the index of icon removed.

The following VB clears the control's icons collection:

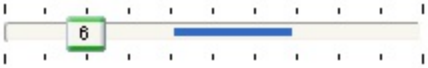
ExSlider1.ReplaceIcon 0, -1

property Slider.SelectRange as Boolean

Returns or sets a value that indicates whether the control can have a select range.

Type	Description
Boolean	A Boolen expression that indicates whether the control can have a select range.

By default, the SelectRange property is False. The SelStart property specifies where the selection starts. The SelLength property specifies the length of the selection. Use the Background property to change the visual aspect of the selection range. Use the Value property to specify the control's value. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The slider control has a capability that you might ignore: you can assign its SelectRange property to True to enter Select Range mode, during which the user can use the Slider to select a *range* instead of a single value. When in Select Range mode, however, it's up to you to manage the SelStart and SelLength properties.



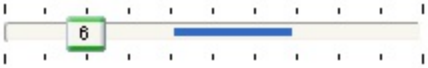
The blue area indicates the selection range. Using the Background property you can specify a new visual appearance of the selection range, including skins. Use the SelStart and SelLenght property to specify the area being occupied by selection range.

property Slider.SelLength as Long

Returns or sets the length of a selection.

Type	Description
Long	A long expression that indicates the length of the selection.

By default, the SelLength property is 0. The SelectRange property returns or sets a value that indicates whether the control can have a select range. The SelStart property returns or sets a value that indicates where a selection starts. Use the Background property to change the visual aspect of the selection range. Use the Value property to specify the control's value. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The slider control has a capability that you might ignore: you can assign its SelectRange property to True to enter Select Range mode, during which the user can use the Slider to select a *range* instead of a single value. When in Select Range mode, however, it's up to you to manage the SelStart and SelLength properties.



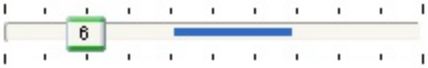
The blue area indicates the selection range. Using the Background property you can specify a new visual appearance of the selection range, including skins. Use the SelStart and SelLenght property to specify the area being occupied by selection range.

property Slider.SelStart as Long

Returns or sets a value that indicates where a selection starts.

Type	Description
Long	A long expression that specifies where the selection starts.

By default, the SelStart property is 0. The SelectRange property returns or sets a value that indicates whether the control can have a select range. The SelLength property specifies the length of the selection. Use the Background property to change the visual aspect of the selection range. Use the Value property to specify the control's value. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The slider control has a capability that you might ignore: you can assign its SelectRange property to True to enter Select Range mode, during which the user can use the Slider to select a *range* instead of a single value. When in Select Range mode, however, it's up to you to manage the SelStart and SelLength properties.



The blue area indicates the selection range. Using the Background property you can specify a new visual appearance of the selection range, including skins. Use the SelStart and SelLenght property to specify the area being occupied by selection range.

property Slider.ShowFocusRect as Boolean

Sets or gets a value that indicates whether the control is marked when it gets the focus.

Type	Description
Boolean	A boolean expression that indicates whether the control is highlighted when the control gets the focus.

By default, the ShowFocusRect property is False. Use the ShowFocusRect property to mark the control that has the focus.

property Slider.ShowImageList as Boolean

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

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

By default, the ShowImageList property is True. Use the ShowImageList property to hide the control's images list window. The control's images list window is visible only at design time. Use the Images method to associate an images list control to the Slider control. Use the ReplaceIcon method to add, remove or clear icons in the control's images collection. Use the HTMLPicture property to display custom size picture in any part of the control.

property Slider.ShowThumbProgress as Boolean

Specifies whether the thumb indicates a progress bar.

Type	Description
Boolean	A Boolean expression that specifies whether the control shows the progressbar instead the thumb or slider.

By default, the ShowThumbProgress property is False. Use the ShowThumbProgress property to change your slider control to a progress bar control. Use the Value property to specify the control's value. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. Use the Background property to change the visual appearance for any part of the control, in any state.

method Slider.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	The ToolTip parameter can be any of the following: • NULL(BSTR) or "<null>"(string) to indicate that the tooltip for the object being hovered is not changed • A String expression that indicates the description of the tooltip, that supports built-in HTML format (adds, replaces or changes the object's tooltip)
Title as Variant	The Title parameter can be any of the following: • missing (VT_EMPTY, VT_ERROR type) or "<null>" (string) the title for the object being hovered is not changed. • A String expression that indicates the title of the tooltip (no built-in HTML format) (adds, replaces or changes the object's title)
Alignment as Variant	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. The Alignment parameter can be one of the following: • 0 - exTopLeft • 1 - exTopRight • 2 - exBottomLeft • 3 - exBottomRight • 0x10 - exCenter • 0x11 - exCenterLeft • 0x12 - exCenterRight • 0x13 - exCenterTop • 0x14 - exCenterBottom By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).

X as Variant	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)
Y 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)

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:

- ** ... ** displays the text in **bold**
- ** ... ** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** ~~Strike~~-through text
- **<a id;options> ... ** 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.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<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
- **
** 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 **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € 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 bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

gradient-center

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

height of the font. For instance the "<**out** 000000><fgcolor=FFFFFF>outlined</fgcolor></**out**>" generates the following picture:

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

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

property Slider.SmallChange as Long

Gets or sets the value added to or subtracted from the Value property when the slider is moved a small distance.

Type	Description
Long	A long expression that indicates the value added to or subtracted from the Value property when the thumb is moved a small distance.

By default, the SmallChange property is 1. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance. If the SmallChange property is 0, the Value property is not changed when clicking the left/up or down/right buttons of the control. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The LabelTick property indicates the HTML expression to be displayed as labels for each tick.

The Value property goes from:

- Minimum to Maximum values

property Slider.SmallChangeF as Double

Gets or sets the value added to or subtracted from the Value property when the slider is moved a small distance(as float).

Type	Description
Double	A floating expression that indicates the value added to or subtracted from the Value property when the thumb is moved a small distance.

By default, the SmallChangeF property is 1.00 The SmallChange property gets or sets the value added to or subtracted from the ValueF property when the thumb is moved a small distance and the AllowFloat property is True. If the SmallChange property is 0, the Value property is not changed when clicking the left/up or down/right buttons of the control. The LargeChangeF property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. Use the MinimumF and MaximumF properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The LabelTick property indicates the HTML expression to be displayed as labels for each tick.

The ValueF property goes from:

- MinimumF to MaximumF values

property Slider.Template as String

Specifies the control's template.

Type	Description
String	A string expression that indicates 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 gets the result after executing a template script.

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 Slider.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.Activex1.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 Slider.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 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.*

property Slider.ThumbSize as Long

Specifies the width or the height of the thumb.

Type	Description
Long	A long expression that defines the size of the control's thumb.

By default, the ThumbSize property is -1. If the ThumbSize property is -1, the control automatically computes its size based on Maximum, Minimum, LargeChange and related properties. If the ThumbSize property is greater than 0, it indicates in pixels the size of the thumb. Use the Orientation property to specify whether the control is vertically or horizontally oriented.

property Slider.TickColor as Color

Specifies the color for the control's ticks.

Type	Description
Color	A Color expression that indicates the color to paint the ticks.

Use the TickColor property specifies the color to paint the ticks. Use the TickStyle property to indicate where the ticks are displayed. The TickFrequency indicates the ratio of ticks in the control.

property Slider.TickFrequency as Long

Returns or sets a value that indicates the ratio of ticks on the control.

Type	Description
Long	A long expression that that indicates the ratio of ticks on the control.

By default, the TickFrequency property is 1. The TickFrequency indicates the ratio of ticks in the control. Use the TickStyle property to indicate where the ticks are displayed. Use the TickColor property specifies the color to paint the ticks. The LabelTick property indicates the HTML expression to be displayed as labels for each tick. The SmallChange property indicates the minimum movement of the slider's position.

property Slider.TickFrequencyF as Double

Returns or sets a value that indicates the ratio of ticks on the control.

Type	Description
Double	A floating expression that that indicates the ratio of ticks on the control.

By default, the TickFrequencyF property is 1.00 The TickFrequencyF indicates the ratio of ticks in the control, when the AllowFloat property is True . Use the TickStyle property to indicate where the ticks are displayed. Use the TickColor property specifies the color to paint the ticks. The LabelTick property indicates the HTML expression to be displayed as labels for each tick. The SmallChangeF property indicates the minimum movement of the slider's position.

property Slider.TickStyle as TickStyleEnum

Specifies where the ticks appears on the control.

Type	Description
TickStyleEnum	A TickStyleEnum expression that indicates where the ticks are displayed

By default, the TickStyle property is exBottomRight. Use the TickStyle property to indicate where the ticks are displayed. The TickFrequency indicates the ratio of ticks in the control. Use the TickColor property specifies the color to paint the ticks.

property Slider.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

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

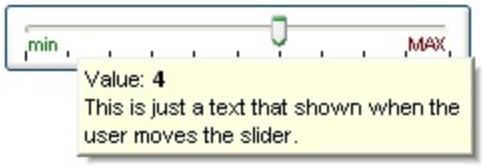
Use the ToolTipFont property to assign a font for the control's tooltip.

property Slider.ToolTipText as String

Specifies the control's tooltip text.

Type	Description
String	A String expression that specifies the tooltip being displayed when the user clicks and moves the control's thumb.

Use the ToolTipText property to assign a tooltip to be displayed when the user clicks and moves the thumb part of the control. Use the ToolTipTitle property to assign a title for the tooltip. The tooltip shows up only when the user clicks and moves the thumb, and the ToolTipText or ToolTipTitle property is not empty. Use the Value property to specify the control's value. Use the Minimum and Maximum properties to specify the range's value. The control fires the Change event property when the user changes the position of the thumb.



The following VB sample displays a tooltip when user moves the thumb:

```
Private Sub Slider1_Change()
   With Slider1
      .Object.ToolTipText = "Record " & .Value & "/" & .Maximum
      .ToolTipTitle = "Position"
   End With
End Sub
```

The following VB/NET sample displays a tooltip when user moves the thumb:

```
Private Sub AxSlider1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxSlider1.Change
   With AxSlider1
      .ToolTipText = "Record " & .Value.ToString() & "/" & .Maximum.ToString()
      .ToolTipTitle = "Position"
   End With
End Sub
```

The following C# sample displays a tooltip when user moves the thumb:

```
private void axSlider1_Change(object sender, EventArgs e)
{
   axSlider1.ToolTipText = "Record " + axSlider1.Value.ToString() + "/" +
axSlider1.Maximum.ToString();
   axSlider1.ToolTipTitle = "Position";
}
```

The following C++ sample displays a tooltip when user moves the thumb:

```
void OnChangeSlider1()
{
   CString strFormat;
   strFormat.Format( _T("Record %i/%i"), m_slider.GetValue(), m_slider.GetMaximum() );
   m_slider.SetToolTipText( strFormat );
   m_slider.SetToolTipTitle( "Position" );
}
```

The following VFP sample displays a tooltip when user moves the thumb:

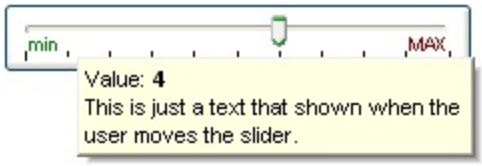
```
*** ActiveX Control Event ***

with thisform.Slider1
   .Object.ToolTipText = "Record " + ltrim(str(.Value)) + "/" + ltrim(str(.Maximum))
   .ToolTipTitle = "Position"
endwith
```

property Slider.ToolTipTitle as String

Specifies the title of the control's tooltip.

Type	Description
String	A String expression that specifies the title of the tooltip being displayed when the user clicks and moves the control's thumb.

Use the ToolTipTitle property to assign a title for the tooltip. Use the ToolTipText property to assign a tooltip to be displayed when the user clicks and moves the thumb part of the control. The tooltip shows up only when the user clicks and moves the thumb, and the ToolTipText or ToolTipTitle property is not empty. Use the Value property to specify the control's value. Use the Minimum and Maximum properties to specify the range's value. The control fires the Change event property when the user changes the position of the thumb.



The following VB sample displays a tooltip when user moves the thumb:

```
Private Sub Slider1_Change()
   With Slider1
      .Object.ToolTipText = "Record " & .Value & "/" & .Maximum
      .ToolTipTitle = "Position"
   End With
End Sub
```

The following VB/NET sample displays a tooltip when user moves the thumb:

```
Private Sub AxSlider1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxSlider1.Change
   With AxSlider1
      .ToolTipText = "Record " & .Value.ToString() & "/" & .Maximum.ToString()
      .ToolTipTitle = "Position"
   End With
End Sub
```

The following C# sample displays a tooltip when user moves the thumb:

```
private void axSlider1_Change(object sender, EventArgs e)
{
   axSlider1.ToolTipText = "Record " + axSlider1.Value.ToString() + "/" +
axSlider1.Maximum.ToString();
   axSlider1.ToolTipTitle = "Position";
}
```

The following C++ sample displays a tooltip when user moves the thumb:

```
void OnChangeSlider1()
{
   CString strFormat;
   strFormat.Format( _T("Record %i/%i"), m_slider.GetValue(), m_slider.GetMaximum() );
   m_slider.SetToolTipText( strFormat );
   m_slider.SetToolTipTitle( "Position" );
}
```

The following VFP sample displays a tooltip when user moves the thumb:

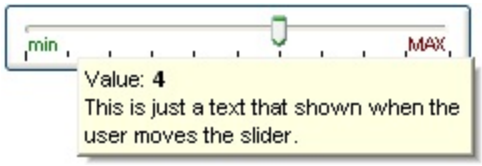
```
*** ActiveX Control Event ***

with thisform.Slider1
   .Object.ToolTipText = "Record " + ltrim(str(.Value)) + "/" + ltrim(str(.Maximum))
   .ToolTipTitle = "Position"
endwith
```

property Slider.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.



property Slider.ToolTipX as String

Indicates an expression that determines the horizontal-position of the tooltip, in screen coordinates.

Type	Description
String	Specifies the expression to show slider's horizontal-position of the tooltip

property Slider.ToolTipY as String

Indicates an expression that determines the vertical-position of the tooltip, in screen coordinates.

Type	Description
String	Specifies the expression to show slider's vertical-position of the tooltip

property Slider.UserData(Part as PartEnum) as Variant

Associates an extra data to a part of the control.

Type	Description
Part as PartEnum	A PartEnum expression that specifies the part to assign an extra data.
Variant	A Variant expression that indicates the extra data being assigned to a part of the control.

use the UserData property to assign an extra data to a part of the control. Use the Caption property to specify the part's caption. Use the Background property to change the visual appearance of any part of the control. Use the VisiblePart or VisibleParts property to specify visible parts in the control. Use the EnablePart or EnableParts property to specify which parts are enabled or disabled. Use the OwnerDrawPart property to specify an owner draw part.

property Slider.Value as Long

The value that the slider position represents.

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

The trial/demo version of the control always retrieves an arbitrary (random) value. The registered version of the control retrieves the correctly value.

The Value property specifies the control's value. The control fires the Change event after user changes the control's value. The control fires the Changing property before changing the control's value. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. Use the Background property to change the visual appearance for any part of the control, in any state.

The Value property goes from:

- Minimum to Maximum values

For instance, the following VB sample prints the control's Value on the control's thumb:

```
Private Sub Slider1_Change()
   With Slider1
      .Caption(exThumbPart) = .Value
   End With
End Sub
```

The following C++ sample prints the control's Value on the control's thumb:

```
void OnChangeSlider1()
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_slider.GetValue() );
   m_slider.SetCaption( 256, strFormat );
}
```

The following VB.NET sample prints the control's Value on the control's thumb:

```
With AxSlider1
   .set_Caption(EXSLIDERLib.PartEnum.exThumbPart, .Value.ToString())
End With
```

The following C# sample prints the control's Value on the control's thumb:

```
private void axSlider1_Change(object sender, EventArgs e)
{
   axSlider1.set_Caption(EXSLIDERLib.PartEnum.exThumbPart, axSlider1.Value.ToString());
}
```

The following VFP sample prints the control's Value on the control's thumb:

```
*** ActiveX Control Event ***

with thisform.Slider1
   .Caption(256) = .Value
endwith
```

property Slider.ValueF as Double

The value that the thumb box position represents (as float)(as float).

Type	Description
Double	A floating expression that indicates the control's value.

The trial/demo version of the control always retrieves an arbitrary (random) value. The registered version of the control retrieves the correctly value.

The ValueF property specifies the control's value. The ValueF property has effect ONLY if the AllowFloat property is True. The control fires the Change event after user changes the control's value. The control fires the Changing property before changing the control's value. Use the MinimumF and MaximumF properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The SmallChangeF property gets or sets the value added to or subtracted from the ValueF property when the thumb is moved a small distance. The LargeChangeF property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance. Use the Background property to change the visual appearance for any part of the control, in any state. The LabelTick property indicates the HTML expression to be displayed as labels for each tick.

The ValueF property goes from:

- MinimumF to MaximumF values

For instance, the following samples print the control's ValueF on the control's thumb:

VBA (MS Access, Excell...)

```
' Change event - Occurs when the value of the control is changed.
Private Sub Slider1_Change()
   With Slider1
      .Caption(256) = .ValueF
   End With
End Sub

With Slider1
   .BeginUpdate
   .AllowFloat = True
```

```
    .MinimumF = -3.25
    .MaximumF = 3.25
    .SmallChangeF = 0.25
    .ThumbSize = 48
    .ValueF = 0
    .TickStyle = 2
    .TickFrequencyF = 0
    .EndUpdate
End With
```

VB6

```
' Change event - Occurs when the value of the control is changed.
Private Sub Slider1_Change()
    With Slider1
        .Caption(exThumbPart) = .ValueF
    End With
End Sub

With Slider1
    .BeginUpdate
    .AllowFloat = True
    .MinimumF = -3.25
    .MaximumF = 3.25
    .SmallChangeF = 0.25
    .ThumbSize = 48
    .ValueF = 0
    .TickStyle = exBoth
    .TickFrequencyF = 0
    .EndUpdate
End With
```

VB.NET

```
' Change event - Occurs when the value of the control is changed.
Private Sub Exslider1_Change(ByVal sender As System.Object) Handles
Exslider1.Change
    With Exslider1
```

```
        .set_Caption(exontrol.EXSLIDERLib.PartEnum.exThumbPart,.ValueF)
    End With
End Sub

With Exslider1
    .BeginUpdate()
    .AllowFloat = True
    .MinimumF = -3.25
    .MaximumF = 3.25
    .SmallChangeF = 0.25
    .ThumbSize = 48
    .ValueF = 0
    .TickStyle = exontrol.EXSLIDERLib.TickStyleEnum.exBoth
    .TickFrequencyF = 0
    .EndUpdate()
End With
```

VB.NET for /COM

```
' Change event - Occurs when the value of the control is changed.
Private Sub AxSlider1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxSlider1.Change
    With AxSlider1
        .set_Caption(EXSLIDERLib.PartEnum.exThumbPart,.ValueF)
    End With
End Sub

With AxSlider1
    .BeginUpdate()
    .AllowFloat = True
    .MinimumF = -3.25
    .MaximumF = 3.25
    .SmallChangeF = 0.25
    .ThumbSize = 48
    .ValueF = 0
    .TickStyle = EXSLIDERLib.TickStyleEnum.exBoth
    .TickFrequencyF = 0
```

```
    .EndUpdate()
End With
```

C++

```
// Change event - Occurs when the value of the control is changed.
void OnChangeSlider1()
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXSLIDERLib' for the library: 'ExSlider 1.0 Control
Library'
        #import <ExSlider.dll>
        using namespace EXSLIDERLib;
    */
    EXSLIDERLib::ISliderPtr spSlider1 = GetDlgItem(IDC_SLIDER1)-
>GetControlUnknown();
    spSlider1->PutCaption(EXSLIDERLib::exThumbPart,_bstr_t(spSlider1-
>GetValueF()));
}

EXSLIDERLib::ISliderPtr spSlider1 = GetDlgItem(IDC_SLIDER1)-
>GetControlUnknown();
spSlider1->BeginUpdate();
spSlider1->PutAllowFloat(VARIANT_TRUE);
spSlider1->PutMinimumF(-3.25);
spSlider1->PutMaximumF(3.25);
spSlider1->PutSmallChangeF(0.25);
spSlider1->PutThumbSize(48);
spSlider1->PutValueF(0);
spSlider1->PutTickStyle(EXSLIDERLib::exBoth);
spSlider1->PutTickFrequencyF(0);
spSlider1->EndUpdate();
```

C++ Builder

```
// Change event - Occurs when the value of the control is changed.
```

```
void __fastcall TForm1::Slider1Change(TObject *Sender)
{
   Slider1->Caption[Exsliderlib_tlb::PartEnum::exThumbPart] = PChar(Slider1-
>ValueF);
}

Slider1->BeginUpdate();
Slider1->AllowFloat = true;
Slider1->MinimumF = -3.25;
Slider1->MaximumF = 3.25;
Slider1->SmallChangeF = 0.25;
Slider1->ThumbSize = 48;
Slider1->ValueF = 0;
Slider1->TickStyle = Exsliderlib_tlb::TickStyleEnum::exBoth;
Slider1->TickFrequencyF = 0;
Slider1->EndUpdate();
```

C#

```
// Change event - Occurs when the value of the control is changed.
private void exslider1_Change(object sender)
{

exslider1.set_Caption(exontrol.EXSLIDERLib.PartEnum.exThumbPart,exslider1.ValueF.ToS

}
//this.exslider1.Change += new
exontrol.EXSLIDERLib.exg2antt.ChangeEventHandler(this.exslider1_Change);

exslider1.BeginUpdate();
exslider1.AllowFloat = true;
exslider1.MinimumF = -3.25;
exslider1.MaximumF = 3.25;
exslider1.SmallChangeF = 0.25;
exslider1.ThumbSize = 48;
exslider1.ValueF = 0;
```

```
exslider1.TickStyle = exontrol.EXSLIDERLib.TickStyleEnum.exBoth;
exslider1.TickFrequencyF = 0;
exslider1.EndUpdate();
```

JavaScript

```
<SCRIPT FOR="Slider1" EVENT="Change()" LANGUAGE="JScript">
   Slider1.Caption(256) = Slider1.ValueF;
</SCRIPT>

<OBJECT classid="clsid:031F9B36-1219-4DF5-8E09-1A50B8185BC2" id="Slider1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   Slider1.BeginUpdate();
   Slider1.AllowFloat = true;
   Slider1.MinimumF = -3.25;
   Slider1.MaximumF = 3.25;
   Slider1.SmallChangeF = 0.25;
   Slider1.ThumbSize = 48;
   Slider1.ValueF = 0;
   Slider1.TickStyle = 2;
   Slider1.TickFrequencyF = 0;
   Slider1.EndUpdate();
</SCRIPT>
```

C# for /COM

```
// Change event - Occurs when the value of the control is changed.
private void axSlider1_Change(object sender, EventArgs e)
{

axSlider1.set_Caption(EXSLIDERLib.PartEnum.exThumbPart,axSlider1.ValueF.ToString());

}
//this.axSlider1.Change += new EventHandler(this.axSlider1_Change);
```

```
axSlider1.BeginUpdate();
axSlider1.AllowFloat = true;
axSlider1.MinimumF = -3.25;
axSlider1.MaximumF = 3.25;
axSlider1.SmallChangeF = 0.25;
axSlider1.ThumbSize = 48;
axSlider1.ValueF = 0;
axSlider1.TickStyle = EXSLIDERLib.TickStyleEnum.exBoth;
axSlider1.TickFrequencyF = 0;
axSlider1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
// Change event - Occurs when the value of the control is changed.
void onEvent_Change()
{
    ;
    exslider1.Caption(256/*exThumbPart*/,exslider1.ValueF());
}

public void init()
{
    ;

    super();

    exslider1.BeginUpdate();
    exslider1.AllowFloat(true);
    exslider1.MinimumF(-3.25);
    exslider1.MaximumF(3.25);
    exslider1.SmallChangeF(0.25);
    exslider1.ThumbSize(48);
    exslider1.ValueF(0);
    exslider1.TickStyle(2/*exBoth*/);
    exslider1.TickFrequencyF(0);
    exslider1.EndUpdate();
```

```
}
```

Delphi 8 (.NET only)

```
// Change event - Occurs when the value of the control is changed.
procedure TWinForm1.AxSlider1_Change(sender: System.Object; e:
System.EventArgs);
begin
   with AxSlider1 do
   begin
      set_Caption(EXSLIDERLib.PartEnum.exThumbPart,ValueF);
   end
end;

with AxSlider1 do
begin
   BeginUpdate();
   AllowFloat := True;
   MinimumF := -3.25;
   MaximumF := 3.25;
   SmallChangeF := 0.25;
   ThumbSize := 48;
   ValueF := 0;
   TickStyle := EXSLIDERLib.TickStyleEnum.exBoth;
   TickFrequencyF := 0;
   EndUpdate();
end
```

Delphi (standard)

```
// Change event - Occurs when the value of the control is changed.
procedure TForm1.Slider1Change(ASender: TObject; );
begin
   with Slider1 do
   begin
      Caption[EXSLIDERLib_TLB.exThumbPart] := ValueF;
   end
end;
```

```
with Slider1 do
begin
   BeginUpdate();
   AllowFloat := True;
   MinimumF := -3.25;
   MaximumF := 3.25;
   SmallChangeF := 0.25;
   ThumbSize := 48;
   ValueF := 0;
   TickStyle := EXSLIDERLib_TLB.exBoth;
   TickFrequencyF := 0;
   EndUpdate();
end
```

VFP

```
*** Change event - Occurs when the value of the control is changed. ***
LPARAMETERS nop
   with thisform.Slider1
      .Caption(256) = .ValueF
   endwith

with thisform.Slider1
   .BeginUpdate
   .AllowFloat = .T.
   .MinimumF = -3.25
   .MaximumF = 3.25
   .SmallChangeF = 0.25
   .ThumbSize = 48
   .ValueF = 0
   .TickStyle = 2
   .TickFrequencyF = 0
   .EndUpdate
endwith
```

dBASE Plus

```
/*
with (this.ACTIVEX1.nativeObject)
   Change = class::nativeObject_Change
endwith
*/
// Occurs when the value of the control is changed.
function nativeObject_Change()
   local oSlider
   oSlider = form.Activex1.nativeObject
   oSlider.Template = [Caption(256) = Str(ValueF)] // oSlider.Caption(256) =
Str(oSlider.ValueF)
return

local oSlider

oSlider = form.Activex1.nativeObject
oSlider.BeginUpdate()
oSlider.AllowFloat = true
oSlider.MinimumF = -3.25
oSlider.MaximumF = 3.25
oSlider.SmallChangeF = 0.25
oSlider.ThumbSize = 48
oSlider.ValueF = 0
oSlider.TickStyle = 2
oSlider.TickFrequencyF = 0
oSlider.EndUpdate()
```

Visual Objects

```
METHOD OCX_Exontrol1Change() CLASS MainDialog
   // Change event - Occurs when the value of the control is changed.
   oDCOCX_Exontrol1:[Caption,exThumbPart] := AsString(oDCOCX_Exontrol1:ValueF)
RETURN NIL



oDCOCX_Exontrol1:BeginUpdate()
```

```
oDCOCX_Exontrol1:AllowFloat := true
oDCOCX_Exontrol1:MinimumF := -3.25
oDCOCX_Exontrol1:MaximumF := 3.25
oDCOCX_Exontrol1:SmallChangeF := 0.25
oDCOCX_Exontrol1:ThumbSize := 48
oDCOCX_Exontrol1:ValueF := 0
oDCOCX_Exontrol1:TickStyle := exBoth
oDCOCX_Exontrol1:TickFrequencyF := 0
oDCOCX_Exontrol1:EndUpdate()
```

PowerBuilder

```
/*begin event Change() - Occurs when the value of the control is changed.*/
/*
   OleObject oSlider
   oSlider = ole_1.Object
   oSlider.Caption(256,String(oSlider.ValueF))
*/
/*end event Change*/

OleObject oSlider

oSlider = ole_1.Object
oSlider.BeginUpdate()
oSlider.AllowFloat = true
oSlider.MinimumF = -3.25
oSlider.MaximumF = 3.25
oSlider.SmallChangeF = 0.25
oSlider.ThumbSize = 48
oSlider.ValueF = 0
oSlider.TickStyle = 2
oSlider.TickFrequencyF = 0
oSlider.EndUpdate()
```

property Slider.ValueFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as Long

Retrieves the value from the point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Long	A long expression that indicates the value from the point.

The trial/demo version of the control always retrieves an arbitrary (random) value. The registered version of the control retrieves the correctly value.

Use the ValueFromPoint property to determine the value from the cursor. The PartFromPoint property specifies the part of the control from the cursor. Use the VisiblePart or VisibleParts property to specify the visible parts of the control.

The following VB sample jumps to the value from the point when the user clicks the button:

```
Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Slider1.Value = Slider1.ValueFromPoint(-1, -1)
End Sub
```

The following VB.NET sample jumps to the value from the point when the user clicks the button:

```
Private Sub AxSlider1_MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXSLIDERLib._ISliderEvents_MouseDownEvent) Handles AxSlider1.MouseDownEvent
   AxSlider1.Value = AxSlider1.get_ValueFromPoint(-1, -1)
End Sub
```

The following C++ sample jumps to the value from the point when the user clicks the button:

```
void OnMouseDownSlider1(short Button, short Shift, long X, long Y)
{
   m_slider.SetValue( m_slider.GetValueFromPoint(-1,-1) );
```

```
}
```

The following C# sample jumps to the value from the point when the user clicks the button:

```
private void axSlider1_MouseDownEvent(object sender,
AxEXSLIDERLib._ISliderEvents_MouseDownEvent e)
{
    axSlider1.Value = axSlider1.get_ValueFromPoint(-1, -1);
}
```

The following VFP sample jumps to the value from the point when the user clicks the button:

```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform.slider1
    .Value = .ValueFromPoint(-1,-1)
endwith
```

property Slider.ValueFromPointF (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as Double

Retrieves the value from the point (as float).

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Double	A long expression that indicates the value from the point.

The trial/demo version of the control always retrieves an arbitrary (random) value. The registered version of the control retrieves the correctly value.

Use the ValueFromPointF property to determine the value from the cursor. This property has effect only if the AllowFloat property it True. The PartFromPoint property specifies the part of the control from the cursor. Use the VisiblePart or VisibleParts property to specify the visible parts of the control.

The following VB sample jumps to the value from the point when the user clicks the button:

```
Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Slider1.Value = Slider1.ValueFromPoint(-1, -1)
End Sub
```

The following VB.NET sample jumps to the value from the point when the user clicks the button:

```
Private Sub AxSlider1_MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXSLIDERLib._ISliderEvents_MouseDownEvent) Handles AxSlider1.MouseDownEvent
   AxSlider1.Value = AxSlider1.get_ValueFromPoint(-1, -1)
End Sub
```

The following C++ sample jumps to the value from the point when the user clicks the button:

```
void OnMouseDownSlider1(short Button, short Shift, long X, long Y)
{
```

```
    m_slider.SetValue( m_slider.GetValueFromPoint(-1,-1) );
}
```

The following C# sample jumps to the value from the point when the user clicks the button:

```
private void axSlider1_MouseDownEvent(object sender,
AxEXSLIDERLib._ISliderEvents_MouseDownEvent e)
{
    axSlider1.Value = axSlider1.get_ValueFromPoint(-1, -1);
}
```

The following VFP sample jumps to the value from the point when the user clicks the button:

```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform.slider1
    .Value = .ValueFromPoint(-1,-1)
endwith
```

property Slider.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 control's version.

property Slider.VisiblePart(Part as PartEnum) as Boolean

Indicates whether the specified part is visible or hidden.

Type	Description
Part as PartEnum	A PartEnum expression or a combination of PartEnum expressions being shown or hidden
Boolean	A boolean expression that indicates whether the part is visible or hidden

The VisiblePart property specifies which part is visible and which part is hidden. The VisibleParts property is similar to VisiblePart property, excepts that all parts must be specified. By default, when a part becomes visible, the EnablePart property is automatically called, so it becomes enabled. The control fires the ClickPart event when the user clicks a part of the control. The ClickingPart event is fired continuously while the user keeps clicking the part of the control. Use the Background property to specify a visual appearance for a specified part of the control in a certain state.

By default, the following parts are shown:

- exLowerBackPart (the part between the start and the thumb part of the control)
- exThumbPart (the thumb/slider part)
- exUpperBackPart (the part between the thumb and the end part of the control)

property Slider.VisibleParts as Long

Specifies the parts of the control being visible.

Type	Description
Long	A long expression that specifies an OR combination of PartEnum values that indicates which parts are visible and which parts are not shown.

By default, the VisibleParts property is 897 (that's a OR combination of exLowerBackPart, exThumbPart, exUpperBackPart). The VisiblePart property specifies which part is visible and which part is hidden. By default, when a part becomes visible, the EnablePart property is automatically called, so it becomes enabled. Use the Background property to specify a visual appearance for a specified part of the control in a certain state.

By default, the following parts are shown:

- exLowerBackPart (the part between the start and the thumb part of the control)
- exThumbPart (the thumb/slider part)
- exUpperBackPart (the part between the thumb and the end part of the control)

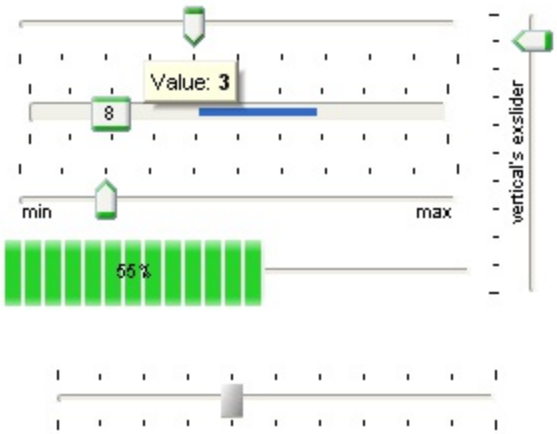
The control fires the ClickPart event when the user clicks a part of the control. The ClickingPart event is fired continuously while the user keeps clicking the part of the control.

property Slider.VisualAppearance as Appearance

Retrieves the control's appearance.

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

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



ExSlider 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: {031F9B36-1219-4DF5-8E09-1A50B8185BC2}. The object's program identifier is: "Exontrol.Slider". The /COM object module is: "ExSlider.dll"

The component supports the following events:

Name	Description
Change	Occurs when the value of the control is changed.
Changing	Occurs when the value of the control is about to change.
Click	Occurs when the user presses and then releases the left mouse button over the control.
ClickingPart	Occurs while the user keeps clicking the part.
ClickPart	Fired when the user clicks a part of the control.
DblClick	Occurs when the user dblclk the left mouse button over an object.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
MouseDown	Occurs when the user presses a mouse button.
MouseMove	Occurs when the user moves the mouse.
MouseUp	Occurs when the user releases a mouse button.
OwnerDrawEnd	Ends painting the owner draw part.
OwnerDrawStart	Starts painting the owner draw part.

event Change ()

Occurs when the value of the control is changed.

Type	Description

Use the Change event to notify your application when the control's Value/ValueF is changed. The Value property of the control specifies the value of the control. Use the Minimum/MinimumF and Maximum/MaximumF properties to specify the range's value. The control fires Changing event just before changing the control's value. Use the Caption property to put a HTML text on any part of the control.

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

C#

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

VB

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

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

C#

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

C++

```
void OnChange()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event Change()
end event Change
```

VB.NET

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

VB6

```
Private Sub Change()
End Sub
```

VBA

```
Private Sub Change()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnChange(oSlider)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComChange
   Forward Send OnComChange
```

```
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_Change()
{
}
```

XBasic

```
function Change as v ()
end function
```

dBASE

```
function nativeObject_Change()
return
```

For instance, the following VB sample prints the control's Value on the control's thumb:

```
Private Sub Slider1_Change()
   With Slider1
      .Caption(exThumbPart) = .Value
   End With
End Sub
```

The following C++ sample prints the control's Value on the control's thumb:

```
void OnChangeSlider1()
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_slider.GetValue() );
   m_slider.SetCaption( 256, strFormat );
}
```

The following VB.NET sample prints the control's Value on the control's thumb:

```
With AxSlider1
   .set_Caption(EXSLIDERLib.PartEnum.exThumbPart, .Value.ToString())
End With
```

The following C# sample prints the control's Value on the control's thumb:

```
private void axSlider1_Change(object sender, EventArgs e)
{
    axSlider1.set_Caption(EXSLIDERLib.PartEnum.exThumbPart, axSlider1.Value.ToString());
}
```

The following VFP sample prints the control's Value on the control's thumb:

```
*** ActiveX Control Event ***

with thisform.Slider1
    .Caption(256) = .Value
endwith
```

event Changing (OldValue as Long, NewValue as Long)

Occurs when the value of the control is about to change.

Type	Description
OldValue as Long	A long expression that indicates the control's Value before performing the change.
NewValue as Long	(by reference) A long expression that indicates the control's newly value.

The trial/demo version of the control always retrieves an arbitrary (random) value for OldValue and NewValue parameters. The registered version of the control retrieves the correctly values.

The Changing event notifies your application just before changing the control's Value. Use the Changing event to prevent specified values, since the NewValue parameter is passed by reference so you can change during the handler. The control fires the Change event after user changes the value. Use the Minimum and Maximum properties to specify the range's value. Use the Caption property to put a HTML text on any part of the control. The SmallChange property gets or sets the value added to or subtracted from the Value property when the thumb is moved a small distance. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the slider is moved a large distance.

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

C#
```
private void Changing(object sender,int OldValue,ref int NewValue)
{
}
```

VB
```
Private Sub Changing(ByVal sender As System.Object,ByVal OldValue As
Integer,ByRef NewValue As Integer) Handles Changing
End Sub
```

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

C#
```
private void Changing(object sender,
AxEXSLIDERLib._ISliderEvents_ChangingEvent e)
{
}
```

C++

```
void OnChanging(long OldValue,long FAR* NewValue)
{
}
```

C++ Builder

```
void __fastcall Changing(TObject *Sender,long OldValue,long * NewValue)
{
}
```

Delphi

```
procedure Changing(ASender: TObject; OldValue : Integer;var NewValue : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure Changing(sender: System.Object; e:
AxEXSLIDERLib._ISliderEvents_ChangingEvent);
begin
end;
```

Powe…

```
begin event Changing(long OldValue,long NewValue)
end event Changing
```

VB.NET

```
Private Sub Changing(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_ChangingEvent) Handles Changing
End Sub
```

VB6

```
Private Sub Changing(ByVal OldValue As Long,NewValue As Long)
End Sub
```

VBA

```
Private Sub Changing(ByVal OldValue As Long,NewValue As Long)
End Sub
```

VFP

```
LPARAMETERS OldValue,NewValue
```

Xbas…

```
PROCEDURE OnChanging(oSlider,OldValue,NewValue)
RETURN
```

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

Java…
```
<SCRIPT EVENT="Changing(OldValue,NewValue)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function Changing(OldValue,NewValue)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComChanging Integer llOldValue Integer llNewValue
   Forward Send OnComChanging llOldValue llNewValue
End_Procedure
```

Visual Objects
```
METHOD OCX_Changing(OldValue,NewValue) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_Changing(int _OldValue,COMVariant /*long*/ _NewValue)
{
}
```

XBasic
```
function Changing as v (OldValue as N,NewValue as N)
end function
```

dBASE
```
function nativeObject_Changing(OldValue,NewValue)
return
```

The following VB sample limits the control's Value to SmallChange property:

```
Private Sub Slider1_Changing(ByVal OldValue As Long, NewValue As Long)
   With Slider1
      NewValue = CLng(NewValue / .SmallChange) * .SmallChange
      If NewValue > .Maximum Then
         NewValue = .Maximum
      End If
   End With
End Sub
```

The following VB samples prints the old and the new value on the thumb part of the control:

```
Private Sub Slider1_Changing(ByVal OldValue As Long, NewValue As Long)
   With Slider1
      .Caption(exThumbPart) = "<img></img>" & OldValue & " - " & NewValue
   End With
End Sub
```

The following VB.NET samples prints the old and the new value on the thumb part of the control:

```
Private Sub AxSlider1_Changing(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_ChangingEvent) Handles AxSlider1.Changing
   With AxSlider1
      .set_Caption(EXSLIDERLib.PartEnum.exThumbPart, "<img></img>" +
e.oldValue.ToString() + " - " + e.newValue.ToString())
   End With
End Sub
```

The following C++ samples prints the old and the new value on the thumb part of the control:

```
void OnChangingSlider1(long OldValue, long FAR* NewValue)
{
   CString strFormat;
   strFormat.Format( _T("<img></img>%i - %i"), OldValue, *NewValue );
   m_slider.SetCaption( 256, strFormat );
}
```

The following C# samples prints the old and the new value on the thumb part of the control:

```
private void axSlider1_Changing(object sender,
AxEXSLIDERLib._ISliderEvents_ChangingEvent e)
{
   axSlider1.set_Caption(EXSLIDERLib.PartEnum.exThumbPart, "<img></img>" +
e.oldValue.ToString()  + " - " + e.newValue.ToString());
}
```

The following VFP samples prints the old and the new value on the thumb part of the control:

```
*** ActiveX Control Event ***
LPARAMETERS oldvalue, newvalue

with thisform.Slider1
    .Caption(256) = "<img></img>" + ltrim(Str(oldvalue)) + " - " + ltrim(Str(newvalue))
endwith
```

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. The ClickPart event notifies your application that the user clicks a part of the control. The ClickingPart event is fired continuously while the user keeps clicking a part of the control. The PartFromPoint property specifies the part of the control from the cursor. Use the ValueFromPoint property to determine the value from the cursor. Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

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

C#

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

VB

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

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

C#

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

C++

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

C++ Builder

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

Delphi

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

```
end;
```

Delphi 8 (.NET only)

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

Powe…

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

VB.NET

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

VB6

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

VBA

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

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnClick(oSlider)
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 ClickingPart (Part as PartEnum)

Occurs while the user keeps clicking the part.

Type	Description
Part as PartEnum	A PartEnum expression being clicked.

The ClickingPart event is fired continuously while the user keeps clicking the part of the control. The ClickPart event is fired when the user clicks and releases the left mouse button over the part of the control. The VisibleParts property is similar to VisiblePart property, excepts that all parts must be specified. By default, when a part becomes visible, the EnablePart property is automatically called, so it becomes enabled. Use the Background property to specify a visual appearance for a specified part of the control in a certain state.

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

C#

```
private void ClickingPart(object sender,exontrol.EXSLIDERLib.PartEnum Part)
{
}
```

VB

```
Private Sub ClickingPart(ByVal sender As System.Object,ByVal Part As
exontrol.EXSLIDERLib.PartEnum) Handles ClickingPart
End Sub
```

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

C#

```
private void ClickingPart(object sender,
AxEXSLIDERLib._ISliderEvents_ClickingPartEvent e)
{
}
```

C++

```
void OnClickingPart(long Part)
{
}
```

C++ Builder

```
void __fastcall ClickingPart(TObject *Sender,Exsliderlib_tlb::PartEnum Part)
{
}
```

Delphi

```
procedure ClickingPart(ASender: TObject; Part : PartEnum);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure ClickingPart(sender: System.Object; e:
AxEXSLIDERLib._ISliderEvents_ClickingPartEvent);
begin
end;
```

Powe…

```
begin event ClickingPart(long Part)
end event ClickingPart
```

VB.NET

```
Private Sub ClickingPart(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_ClickingPartEvent) Handles ClickingPart
End Sub
```

VB6

```
Private Sub ClickingPart(ByVal Part As EXSLIDERLibCtl.PartEnum)
End Sub
```

VBA

```
Private Sub ClickingPart(ByVal Part As Long)
End Sub
```

VFP

```
LPARAMETERS Part
```

Xbas…

```
PROCEDURE OnClickingPart(oSlider,Part)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComClickingPart OLEPartEnum llPart
   Forward Send OnComClickingPart llPart
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_ClickingPart(Part) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ClickingPart(int _Part)
{
}
```

XBasic

```
function ClickingPart as v (Part as OLE::Exontrol.Slider.1::PartEnum)
end function
```

dBASE

```
function nativeObject_ClickingPart(Part)
return
```

event ClickPart (Part as PartEnum)

Fired when the user clicks a part of the control.

Type	Description
Part as PartEnum	A PartEnum expression being clicked.

The ClickPart event notifies your application that the user clicks a part of the control. The ClickPart event is fired only after releasing the mouse. The ClickingPart event is fired continuously while the user keeps clicking a part of the control. The VisiblePart or VisibleParts property specifies the part being visible or hidden.

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

C#

```
private void ClickPart(object sender,exontrol.EXSLIDERLib.PartEnum Part)
{
}
```

VB

```
Private Sub ClickPart(ByVal sender As System.Object,ByVal Part As
exontrol.EXSLIDERLib.PartEnum) Handles ClickPart
End Sub
```

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

C#

```
private void ClickPart(object sender, AxEXSLIDERLib._ISliderEvents_ClickPartEvent
e)
{
}
```

C++

```
void OnClickPart(long Part)
{
}
```

C++ Builder

```
void __fastcall ClickPart(TObject *Sender,Exsliderlib_tlb::PartEnum Part)
{
}
```

Delphi

```
procedure ClickPart(ASender: TObject; Part : PartEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ClickPart(sender: System.Object; e:
AxEXSLIDERLib._ISliderEvents_ClickPartEvent);
begin
end;
```

Powe…

```
begin event ClickPart(long Part)
end event ClickPart
```

VB.NET

```
Private Sub ClickPart(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_ClickPartEvent) Handles ClickPart
End Sub
```

VB6

```
Private Sub ClickPart(ByVal Part As EXSLIDERLibCtl.PartEnum)
End Sub
```

VBA

```
Private Sub ClickPart(ByVal Part As Long)
End Sub
```

VFP

```
LPARAMETERS Part
```

Xbas…

```
PROCEDURE OnClickPart(oSlider,Part)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComClickPart OLEPartEnum llPart
    Forward Send OnComClickPart llPart
End_Procedure
```

Visual Objects

```
METHOD OCX_ClickPart(Part) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ClickPart(int _Part)
{
}
```

XBasic

```
function ClickPart as v (Part as OLE::Exontrol.Slider.1::PartEnum)
end function
```

dBASE

```
function nativeObject_ClickPart(Part)
return
```

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

Occurs when the user dblclk 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 DblClick event is fired when user double clicks the control. The PartFromPoint property specifies the part of the control from the cursor. Use the ValueFromPoint property to determine the value from the cursor. Use the VisibleParts or VisiblePart property to specify which part of the control is visible or hidden.

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

C#

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

VB

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

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

C#

```
private void DblClick(object sender, AxEXSLIDERLib._ISliderEvents_DblClickEvent e)
{
}
```

C++

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

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

```
procedure DblClick(sender: System.Object; e:
AxEXSLIDERLib._ISliderEvents_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
AxEXSLIDERLib._ISliderEvents_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(oSlider,Shift,X,Y)
RETURN
```

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

Java…

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

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function DblClick(Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

The following VB sample displays the part from the cursor:

```
Private Sub Slider1_DblClick(Shift As Integer, X As Single, Y As Single)
   Debug.Print Slider1.PartFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
End Sub
```

The following VB.NET sample displays the part from the cursor:

```
Private Sub AxSlider1_DblClick(ByVal sender As System.Object, ByVal e As AxEXSLIDERLib._ISliderEvents_DblClickEvent) Handles AxSlider1.DblClick
   Debug.WriteLine(AxSlider1.get_PartFromPoint(e.x, e.y).ToString())
End Sub
```

The following C++ sample displays the part from the cursor:

```
void OnDblClickSlider1(short Shift, long X, long Y)
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_slider.GetPartFromPoint(X, Y ) );
   OutputDebugString( strFormat );
}
```

The following C# sample displays the part from the cursor:

```
private void axSlider1_DblClick(object sender, AxEXSLIDERLib._ISliderEvents_DblClickEvent
e)
{
   System.Diagnostics.Debug.WriteLine(axSlider1.get_PartFromPoint(e.x, e.y).ToString());
}
```

The following VB sample displays the part from the cursor:

```
*** ActiveX Control Event ***
LPARAMETERS shift, x, y

wait window nowait ltrim(str(thisform.slider1.PartFromPoint(x,y)))
```

event KeyDown (KeyCode as Integer, Shift as Integer)

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

Type	Description
KeyCode as Integer	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,
AxEXSLIDERLib._ISliderEvents_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:
AxEXSLIDERLib._ISliderEvents_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
AxEXSLIDERLib._ISliderEvents_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(oSlider,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 (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	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,
AxEXSLIDERLib._ISliderEvents_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:
AxEXSLIDERLib._ISliderEvents_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
AxEXSLIDERLib._ISliderEvents_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(oSlider,KeyAscii)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

event KeyUp (KeyCode as Integer, Shift as Integer)

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

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

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

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

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

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

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

C#
```
private void KeyUpEvent(object sender,
AxEXSLIDERLib._ISliderEvents_KeyUpEvent e)
{
}
```

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

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

```
}
```

Delphi

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

Delphi 8 (.NET only)

```
procedure KeyUpEvent(sender: System.Object; e:
AxEXSLIDERLib._ISliderEvents_KeyUpEvent);
begin
end;
```

Powe…

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

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_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(oSlider,KeyCode,Shift)
RETURN
```

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

Java…

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

VBSc…

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

```
</SCRIPT>
```

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

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

Occurs when the user presses 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 DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. The ClickPart event notifies your application that the user clicks a part of the control. The ClickingPart event is fired continuously while the user keeps clicking a part of the control. The PartFromPoint property specifies the part of the control from the cursor. Use the ValueFromPoint property to determine the value from the cursor.

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,
```

```
AxEXSLIDERLib._ISliderEvents_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:
AxEXSLIDERLib._ISliderEvents_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
AxEXSLIDERLib._ISliderEvents_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(oSlider,Button,Shift,X,Y)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

The following VB sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
```

```
Single)
   With Slider1
      If (0 <> (.PartFromPoint(-1, -1) And exBackgroundPart)) Then
         .Value = .ValueFromPoint(-1, -1)
      End If
   End With
End Sub
```

The following VB sample jumps to the value from the point when the user clicks the button:

```
Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   Slider1.Value = Slider1.ValueFromPoint(-1, -1)
End Sub
```

The following VB.NET sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
Private Sub AxSlider1_MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_MouseDownEvent) Handles AxSlider1.MouseDownEvent
   With AxSlider1
      If (0 <> (.get_PartFromPoint(-1, -1) And EXSLIDERLib.PartEnum.exBackgroundPart))
Then
         .Value = .get_ValueFromPoint(-1, -1)
      End If
   End With
End Sub
```

The following VB.NET sample jumps to the value from the point when the user clicks the button:

```
Private Sub AxSlider1_MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_MouseDownEvent) Handles AxSlider1.MouseDownEvent
   AxSlider1.Value = AxSlider1.get_ValueFromPoint(-1, -1)
End Sub
```

The following C++ sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
void OnMouseDownSlider1(short Button, short Shift, long X, long Y)
```

```
{
   if ( m_slider.GetPartFromPoint(-1,-1) & 640 )
      m_slider.SetValue( m_slider.GetValueFromPoint(-1,-1) );
}
```

The following C++ sample jumps to the value from the point when the user clicks the button:

```
void OnMouseDownSlider1(short Button, short Shift, long X, long Y)
{
   m_slider.SetValue( m_slider.GetValueFromPoint(-1,-1) );
}
```

The following C# sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
private void axSlider1_MouseDownEvent(object sender,
AxEXSLIDERLib._ISliderEvents_MouseDownEvent e)
{
   if ( 0 != ( axSlider1.get_PartFromPoint(-1,-1) &
EXSLIDERLib.PartEnum.exBackgroundPart ) )
      axSlider1.Value = axSlider1.get_ValueFromPoint(-1, -1);
}
```

The following C# sample jumps to the value from the point when the user clicks the button:

```
private void axSlider1_MouseDownEvent(object sender,
AxEXSLIDERLib._ISliderEvents_MouseDownEvent e)
{
   axSlider1.Value = axSlider1.get_ValueFromPoint(-1, -1);
}
```

The following VFP sample jumps to the value from the point when the user clicks the upper or lower part of the control:

```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform.slider1
   if ( 0 # bitand( .PartFromPoint(-1,-1), 640) )
```

```
    .Value = .ValueFromPoint(-1,-1)
  endif
endwith
```

The following VFP sample jumps to the value from the point when the user clicks the button:

```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform.slider1
  .Value = .ValueFromPoint(-1,-1)
endwith
```

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 X location of the mouse pointer. The x 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. The ClickPart event notifies your application that the user clicks a part of the control. The ClickingPart event is fired continuously while the user keeps clicking a part of the control. The PartFromPoint property specifies the part of the control from the cursor. Use the ValueFromPoint property to determine the value from the cursor.

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

C#

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

VB

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

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

C#

```
private void MouseMoveEvent(object sender,
AxEXSLIDERLib._ISliderEvents_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:
AxEXSLIDERLib._ISliderEvents_MouseMoveEvent);
begin
end;
```

Powe…

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

VB.NET

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

VB6

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

VBA

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

VFP

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

Xbas…

```
PROCEDURE OnMouseMove(oSlider,Button,Shift,X,Y)
```

```
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

The following VB sample prints the Value from the point:

```
Private Sub Slider1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   Debug.Print Slider1.PartFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
```

```
End Sub
```

The following VB.NET sample prints the Value from the point:

```
Private Sub AxSlider1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_MouseMoveEvent) Handles AxSlider1.MouseMoveEvent
   Debug.WriteLine(AxSlider1.get_PartFromPoint(e.x, e.y).ToString())
End Sub
```

The following C++ sample prints the Value from the point:

```
void OnMouseMoveSlider1(short Button, short Shift, long X, long Y)
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_slider.GetPartFromPoint(X, Y ) );
   OutputDebugString( strFormat );
}
```

The following C# sample prints the Value from the point:

```
private void axSlider1_MouseMoveEvent(object sender,
AxEXSLIDERLib._ISliderEvents_MouseMoveEvent e)
{
   System.Diagnostics.Debug.WriteLine(axSlider1.get_PartFromPoint(e.x, e.y).ToString());
}
```

The following VFP sample prints the Value from the point:

```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

wait window nowait ltrim(str(thisform.slider1.PartFromPoint(x,y)))
```

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 value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. The ClickPart event notifies your application that the user clicks a part of the control. The ClickingPart event is fired continuously while the user keeps clicking a part of the control. The PartFromPoint property specifies the part of the control from the cursor. Use the ValueFromPoint property to determine the value from the cursor.

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,
```

```
AxEXSLIDERLib._ISliderEvents_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:
AxEXSLIDERLib._ISliderEvents_MouseUpEvent);
begin
end;
```

Powe…

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

VB.NET

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

VB6

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

VBA

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

VFP

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

Xbas…

```
PROCEDURE OnMouseUp(oSlider,Button,Shift,X,Y)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

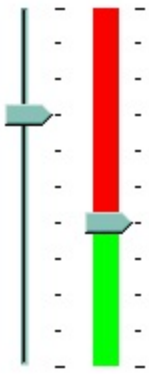
```
function nativeObject_MouseUp(Button,Shift,X,Y)
return
```

event OwnerDrawEnd (Part as PartEnum, hDC as Long)

Ends painting the owner draw part.

Type	Description
Part as PartEnum	A PartEnum expression that indicates the part being painted
hDC as Long	A long expression that indicates the handle to the painting device context (HDC)

The OwnerDrawEnd event occurs after the default painting of the part is done, so it lets the user to paint additional pieces on the default part, if case . The OwnerDrawEnd event is fired only for owner draw parts. Use the OwnerDrawPart property to specify which part is owner draw and which part is not. Use the OwnerDrawStart event to perform painting part before default implementation is called. For instance, if the owner part paints a transparent or lucent skin, the OwnerDrawStart event lets you paint the part before putting the default skin. **The rectangle that should be painted in the device context can be retrieved using the GetClipBox API function**. The VisiblePart or VisibleParts property specifies the part being visible or hidden.



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

C#

```
private void OwnerDrawEnd(object sender,exontrol.EXSLIDERLib.PartEnum Part,int hDC)
{
}
```

VB

```
Private Sub OwnerDrawEnd(ByVal sender As System.Object,ByVal Part As exontrol.EXSLIDERLib.PartEnum,ByVal hDC As Integer) Handles OwnerDrawEnd
End Sub
```

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

C#

```
private void OwnerDrawEnd(object sender,
AxEXSLIDERLib._ISliderEvents_OwnerDrawEndEvent e)
{
}
```

C++

```
void OnOwnerDrawEnd(long Part,long hDC)
{
}
```

C++ Builder

```
void __fastcall OwnerDrawEnd(TObject *Sender,Exsliderlib_tlb::PartEnum Part,long
hDC)
{
}
```

Delphi

```
procedure OwnerDrawEnd(ASender: TObject; Part : PartEnum;hDC : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OwnerDrawEnd(sender: System.Object; e:
AxEXSLIDERLib._ISliderEvents_OwnerDrawEndEvent);
begin
end;
```

Powe…

```
begin event OwnerDrawEnd(long Part,long hDC)
end event OwnerDrawEnd
```

VB.NET

```
Private Sub OwnerDrawEnd(ByVal sender As System.Object, ByVal e As
AxEXSLIDERLib._ISliderEvents_OwnerDrawEndEvent) Handles OwnerDrawEnd
End Sub
```

VB6

```
Private Sub OwnerDrawEnd(ByVal Part As EXSLIDERLibCtl.PartEnum,ByVal hDC As
Long)
End Sub
```

VBA

```
Private Sub OwnerDrawEnd(ByVal Part As Long,ByVal hDC As Long)
End Sub
```

VFP

```
LPARAMETERS Part,hDC
```

Xbas…

```
PROCEDURE OnOwnerDrawEnd(oSlider,Part,hDC)
RETURN
```

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

Java…

```
<SCRIPT EVENT="OwnerDrawEnd(Part,hDC)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OwnerDrawEnd(Part,hDC)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOwnerDrawEnd OLEPartEnum llPart Integer llhDC
   Forward Send OnComOwnerDrawEnd llPart llhDC
End_Procedure
```

Visual Objects

```
METHOD OCX_OwnerDrawEnd(Part,hDC) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OwnerDrawEnd(int _Part,int _hDC)
{
}
```

XBasic

```
function OwnerDrawEnd as v (Part as OLE::Exontrol.Slider.1::PartEnum,hDC as N)
end function
```

dBASE

```
function nativeObject_OwnerDrawEnd(Part,hDC)
return
```

For instance, the following VB sample draws the lower part in red, and the upper part in green (as in the screen shot) :

```
With Slider1
   .OwnerDrawPart(exLowerBackPart Or exUpperBackPart) = True
```

```
End With
```

```
Private Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type
Private Declare Function GetClipBox Lib "gdi32" (ByVal hdc As Long, lpRect As RECT) As Long
Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long
Private Declare Function CreateSolidBrush Lib "gdi32" (ByVal crColor As Long) As Long
Private Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long

Private Sub Slider1_OwnerDrawEnd(ByVal Part As EXSLIDERLibCtl.PartEnum, ByVal hdc As Long)
  Dim r As RECT, h As Long
  GetClipBox hdc, r
  r.Left = r.Left + 4
  r.Right = r.Right - 4
  If Part = exLowerBackPart Then
    h = CreateSolidBrush(RGB(255, 0, 0))
    FillRect hdc, r, h
    DeleteObject (h)
  Else
    If Part = exUpperBackPart Then
      h = CreateSolidBrush(RGB(0, 255, 0))
      FillRect hdc, r, h
      DeleteObject (h)
    End If
  End If
End Sub
```

The following C++ sample draws the lower part in red, and the upper part in green (as in the screen shot) :

```
m_slider.SetOwnerDrawPart( 128 /*exUpperBackPart*/,  TRUE );
```

```
m_slider.SetOwnerDrawPart( 512 /*exLowerBackPart*/,  TRUE );
```

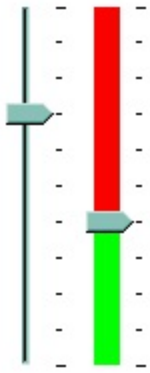
```
void OnOwnerDrawEndSlider1(long Part, long hDC)
{
   HDC h = (HDC)hDC;
   RECT rtPart = {0}; GetClipBox( h, &rtPart );
   InflateRect( &rtPart, -4, 0 );
   switch ( Part )
   {
      case 128: /*exUpperBackPart*/
      {
         HBRUSH hB = CreateSolidBrush( RGB(0,255,0) );
         FillRect( h, &rtPart, hB );
         DeleteObject( hB );
         break;
      }
      case 512: /*exLowerBackPart*/
      {
         HBRUSH hB = CreateSolidBrush( RGB(255,0,0) );
         FillRect( h, &rtPart, hB );
         DeleteObject( hB );
         break;
      }
   }
}
```

event OwnerDrawStart (Part as PartEnum, hDC as Long, DefaultPainting as Boolean)

Starts painting the owner draw part.

Type	Description
Part as PartEnum	A PartEnum expression that indicates the part being painted
hDC as Long	A long expression that indicates the handle to the painting device context (HDC)
DefaultPainting as Boolean	A Boolen expression that indicates whether the default painting should be performed or not. If the DefaultPainting parameter is True, the control paints the part as default, else the part is not painted by the control so the user should draw the entire part.

The OwnerDrawStart event is fired when a part requires to be painted. The OwnerDrawStart event is fired only for owner draw parts. Use the OwnerDrawPart property to specify which part is owner draw and which part is not. You can use the OwnerDrawStart event to avoid painting any part using the DefaultPainting parameter. The control fires the OwnerDrawEnd event when painting the part is done. Use the OwnerDrawStart event to perform painting part before default implementation is called. For instance, if the owner part pains a transparent or lucent skin, the OwnerDrawStart event lets you paint the part before putting the default skin. **The rectangle that should be painted in the device context can be retrieved using the GetClipBox API function**. The VisiblePart or VisibleParts property specifies the part being visible or hidden.



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

C#
```
private void OwnerDrawStart(object sender,exontrol.EXSLIDERLib.PartEnum
Part,int hDC,ref bool DefaultPainting)
{
}
```

VB

```
Private Sub OwnerDrawStart(ByVal sender As System.Object,ByVal Part As exontrol.EXSLIDERLib.PartEnum,ByVal hDC As Integer,ByRef DefaultPainting As Boolean) Handles OwnerDrawStart
End Sub
```

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

C#

```
private void OwnerDrawStart(object sender, AxEXSLIDERLib._ISliderEvents_OwnerDrawStartEvent e)
{
}
```

C++

```
void OnOwnerDrawStart(long Part,long hDC,BOOL FAR* DefaultPainting)
{
}
```

C++ Builder

```
void __fastcall OwnerDrawStart(TObject *Sender,Exsliderlib_tlb::PartEnum Part,long hDC,VARIANT_BOOL * DefaultPainting)
{
}
```

Delphi

```
procedure OwnerDrawStart(ASender: TObject; Part : PartEnum;hDC : Integer;var DefaultPainting : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OwnerDrawStart(sender: System.Object; e: AxEXSLIDERLib._ISliderEvents_OwnerDrawStartEvent);
begin
end;
```

Powe…

```
begin event OwnerDrawStart(long Part,long hDC,boolean DefaultPainting)
end event OwnerDrawStart
```

VB.NET

```
Private Sub OwnerDrawStart(ByVal sender As System.Object, ByVal e As AxEXSLIDERLib._ISliderEvents_OwnerDrawStartEvent) Handles OwnerDrawStart
End Sub
```

VB6

```
Private Sub OwnerDrawStart(ByVal Part As EXSLIDERLibCtl.PartEnum,ByVal hDC As
Long,DefaultPainting As Boolean)
End Sub
```

VBA

```
Private Sub OwnerDrawStart(ByVal Part As Long,ByVal hDC As
Long,DefaultPainting As Boolean)
End Sub
```

VFP

```
LPARAMETERS Part,hDC,DefaultPainting
```

Xbas…

```
PROCEDURE OnOwnerDrawStart(oSlider,Part,hDC,DefaultPainting)
RETURN
```

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

Java…

```
<SCRIPT EVENT="OwnerDrawStart(Part,hDC,DefaultPainting)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OwnerDrawStart(Part,hDC,DefaultPainting)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOwnerDrawStart OLEPartEnum llPart Integer llhDC Boolean
llDefaultPainting
    Forward Send OnComOwnerDrawStart llPart llhDC llDefaultPainting
End_Procedure
```

Visual Objects

```
METHOD OCX_OwnerDrawStart(Part,hDC,DefaultPainting) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OwnerDrawStart(int _Part,int _hDC,COMVariant /*bool*/
_DefaultPainting)
{
```

```
}
```

XBasic

```
function OwnerDrawStart as v (Part as OLE::Exontrol.Slider.1::PartEnum,hDC as
N,DefaultPainting as L)
end function
```

dBASE

```
function nativeObject_OwnerDrawStart(Part,hDC,DefaultPainting)
return
```

For instance, the following VB sample draws the lower part in red, and the upper part in green (as in the screen shot) :

```
With Slider1
   .OwnerDrawPart(exLowerBackPart Or exUpperBackPart) = True
End With
```

```
Private Type RECT
      Left As Long
      Top As Long
      Right As Long
      Bottom As Long
End Type
Private Declare Function GetClipBox Lib "gdi32" (ByVal hdc As Long, lpRect As RECT) As
Long
Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT, ByVal
hBrush As Long) As Long
Private Declare Function CreateSolidBrush Lib "gdi32" (ByVal crColor As Long) As Long
Private Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long

Private Sub Slider1_OwnerDrawEnd(ByVal Part As EXSLIDERLibCtl.PartEnum, ByVal hdc As
Long)
   Dim r As RECT, h As Long
   GetClipBox hdc, r
   r.Left = r.Left + 4
   r.Right = r.Right - 4
   If Part = exLowerBackPart Then
      h = CreateSolidBrush(RGB(255, 0, 0))
```

```
        FillRect hdc, r, h
        DeleteObject (h)
    Else
        If Part = exUpperBackPart Then
            h = CreateSolidBrush(RGB(0, 255, 0))
            FillRect hdc, r, h
            DeleteObject (h)
        End If
    End If
End Sub
```

The following C++ sample draws the lower part in red, and the upper part in green (as in the screen shot) :

```
m_slider.SetOwnerDrawPart( 128 /*exUpperBackPart*/,  TRUE );
m_slider.SetOwnerDrawPart( 512 /*exLowerBackPart*/,  TRUE );
```

```
void OnOwnerDrawEndSlider1(long Part, long hDC)
{
    HDC h = (HDC)hDC;
    RECT rtPart = {0}; GetClipBox( h, &rtPart );
    InflateRect( &rtPart, -4, 0 );
    switch ( Part )
    {
        case 128: /*exUpperBackPart*/
        {
            HBRUSH hB = CreateSolidBrush( RGB(0,255,0) );
            FillRect( h, &rtPart, hB );
            DeleteObject( hB );
            break;
        }
        case 512: /*exLowerBackPart*/
        {
            HBRUSH hB = CreateSolidBrush( RGB(255,0,0) );
            FillRect( h, &rtPart, hB );
            DeleteObject( hB );
            break;
        }
```

```
    }
}
```