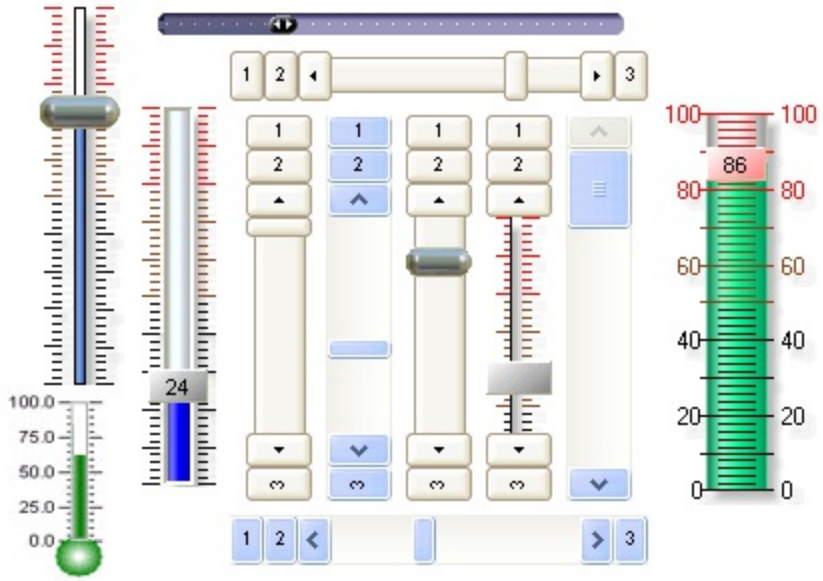
ExScrollBar

The Exontrol's newly exScrollBar control simulates vertical or horizontal scroll bar, slider, track, volume or gauge controls. The exScrollBar 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.

Features include:

- Skinnable Interface support (ability to apply a skin to the any background part)
- Windows XP Theme support
- Owner Draw support
- WYSWYG Template/Layout Editor support
- Ability to have additional buttons above the up/left and down/right arrows
- Multi-lines tooltip support
- ScrollBar or Spin control support
- Ability to put HTML text on any part of the control, includes icons or pictures



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.

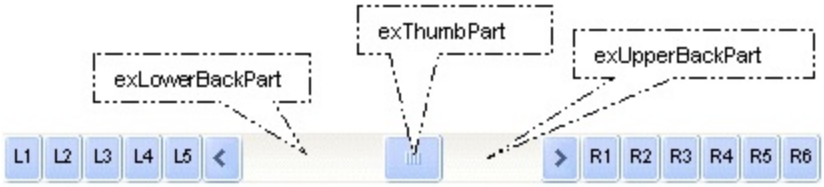
constants AppearanceEnum

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. The following picture shows the parts of the control:



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 scroll bar (Mode property is exVertical).
- All BackgroundPartEnum expressions that starts with **exHS** changes a part in the horizontal scroll bar (Mode 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.
exVSUp	256	Specifies the up button (<) in normal state.

exVSUpP	257	Specifies the up button (<) when it is pressed.
exVSUpD	258	Specifies the up button (<) when it is disabled.
exVSUpH	259	Specifies the up button (<) when the 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.
exVSDown	264	Specifies the down button (>) in normal state.
exVSDownP	265	Specifies the down button (>) when it is pressed.
exVSDownD	266	Specifies the down button (>) when it is disabled.
exVSDownH	267	Specifies the down button (>) when the 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.
		Specifies the background part (exLowerBackPart

exVSBackP	277	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.
exVSUp1	280	Specifies the first additional up button (L1) in normal state.
exVSUp1P	281	Specifies the first additional up button (L1) when it is pressed.
exVSUp1D	282	Specifies the first additional up button (L1) when it is disabled.
exVSUp1H	283	Specifies the first additional up button (L1) when the cursor hovers it.
exVSUp2	284	Specifies the second additional up button (L2) in normal state.
exVSUp2P	285	Specifies the second additional up button (L2) when it is pressed.
exVSUp2D	286	Specifies the second additional up button (L2) when it is disabled.
exVSUp2H	287	Specifies the second additional up button (L2) when the cursor hovers it.
exVSUp3	288	Specifies the third additional up button (L3) in normal state.
exVSUp3P	289	Specifies the third additional up button (L3) when it is pressed.
exVSUp3D	290	Specifies the third additional up button (L3) when it is disabled.
exVSUp3H	291	Specifies the third additional up button (L3) when the cursor hovers it.
exVSUp4	292	Specifies the forth additional up button (L4) in normal state.
exVSUp4P	293	Specifies the forth additional up button (L4) when it is pressed.
exVSUp4D	294	Specifies the forth additional up button (L4) when it is disabled.

exVSUp4H	295	Specifies the forth additional up button (L4) when the cursor hovers it.
exVSUp5	296	Specifies the fifth additional up button (L5) in normal state.
exVSUp5P	297	Specifies the fifth additional up button (L5) when it is pressed.
exVSUp5D	298	Specifies the fifth additional up button (L5) when it is disabled.
exVSUp5H	299	Specifies the fifth additional up button (L5) when the cursor hovers it.
exVSDown1	300	Specifies the first additional down button (R1) in normal state.
exVSDown1P	301	Specifies the first additional down button (R1) when it is pressed.
exVSDown1D	302	Specifies the first additional down button (R1) when it is disabled.
exVSDown1H	303	Specifies the first additional down button (R1) when the cursor hovers it.
exVSDown2	304	Specifies the second additional down button (R2) in normal state.
exVSDown2P	305	Specifies the second additional down button (R2) when it is pressed.
exVSDown2D	306	Specifies the second additional down button (R2) when it is disabled.
exVSDown2H	307	Specifies the second additional down button (R2) when the cursor hovers it.
exVSDown3	308	Specifies the third additional down button (R3) in normal state.
exVSDown3P	309	Specifies the third additional down button (R3) when it is pressed.
exVSDown3D	310	Specifies the third additional down button (R3) when it is disabled.
exVSDown3H	311	Specifies the third additional down button (R3) when the cursor hovers it.
exVSDown4	312	Specifies the forth additional down button (R4) in normal state.

exVSDown4P	313	Specifies the forth additional down button (R4) when it is pressed.
exVSDown4D	314	Specifies the forth additional down button (R4) when it is disabled.
exVSDown4H	315	Specifies the forth additional down button (R4) when the cursor hovers it.
exVSDown5	316	Specifies the fifth additional down button (R5) in normal state.
exVSDown5P	317	Specifies the fifth additional down button (R5) when it is pressed.
exVSDown5D	318	Specifies the fifth additional down button (R5) when it is disabled.
exVSDown5H	319	Specifies the fifth additional down button (R5) when the cursor hovers it..
exVSDown6	320	Specifies the sixth additional down button (R6) in normal state.
exVSDown6P	321	Specifies the sixth additional down button (R6) when it is pressed.
exVSDown6D	322	Specifies the sixth additional down button (R6) when it is disabled.
exVSDown6H	323	Specifies the sixth additional down button (R6) when the cursor hovers it.
exHSLeft	384	Specifies the left button (<) in normal state.
exHSLeftP	385	Specifies the left button (<) when it is pressed.
exHSLeftD	386	Specifies the left button (<) when it is disabled.
exHSLeftH	387	Specifies the left button (<) 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.
exHSRight	392	Specifies the right button (>) in normal state.

exHSRightP	393	Specifies the right button (>) when it is pressed.
exHSRightD	394	Specifies the right button (>) when it is disabled.
exHSRightH	395	Specifies the right button (>) 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.
exHSLeft1	408	Specifies the first additional left button (L1) in normal state.
exHSLeft1P	409	Specifies the first additional left button (L1) when it is pressed.
exHSLeft1D	410	Specifies the first additional left button (L1) when it is disabled.
exHSLeft1H	411	Specifies the first additional left button (L1) when the cursor hovers it.

exHSLeft2	412	Specifies the second additional left button (L2) in normal state.
exHSLeft2P	413	Specifies the second additional left button (L2) when it is pressed.
exHSLeft2D	414	Specifies the second additional left button (L2) when it is disabled.
exHSLeft2H	415	Specifies the second additional left button (L2) when the cursor hovers it.
exHSLeft3	416	Specifies the third additional left button (L3) in normal state.
exHSLeft3P	417	Specifies the third additional left button (L3) when it is pressed.
exHSLeft3D	418	Specifies the third additional left button (L3) when it is disabled.
exHSLeft3H	419	Specifies the third additional left button (L3) when the cursor hovers it.
exHSLeft4	420	Specifies the forth additional left button (L4) in normal state.
exHSLeft4P	421	Specifies the forth additional left button (L4) when it is pressed.
exHSLeft4D	422	Specifies the forth additional left button (L4) when it is disabled.
exHSLeft4H	423	Specifies the forth additional left button (L4) when the cursor hovers it.
exHSLeft5	424	Specifies the fifth additional left button (L5) in normal state.
exHSLeft5P	425	Specifies the fifth additional left button (L5) when it is pressed.
exHSLeft5D	426	Specifies the fifth additional left button (L5) when it is disabled.
exHSLeft5H	427	Specifies the fifth additional left button (L5) when the cursor hovers it.
exHSRight1	428	Specifies the first additional right button (R1) in normal state.
exHSRight1P	429	Specifies the first additional right button (R1) when it is pressed.

exHSRight1D	430	Specifies the first additional right button (R1) when it is disabled.
exHSRight1H	431	Specifies the first additional right button (R1) when the cursor hovers it.
exHSRight2	432	Specifies the second additional right button (R2) in normal state.
exHSRight2P	433	Specifies the second additional right button (R2) when it is pressed.
exHSRight2D	434	Specifies the second additional right button (R2) when it is disabled.
exHSRight2H	435	Specifies the second additional right button (R2) when the cursor hovers it.
exHSRight3	436	Specifies the third additional right button (R3) in normal state.
exHSRight3P	437	Specifies the third additional right button (R3) when it is pressed.
exHSRight3D	438	Specifies the third additional right button (R3) when it is disabled.
exHSRight3H	439	Specifies the third additional right button (R3) when the cursor hovers it.
exHSRight4	440	Specifies the forth additional right button (R4) in normal state.
exHSRight4P	441	Specifies the forth additional right button (R4) when it is pressed.
exHSRight4D	442	Specifies the forth additional right button (R4) when it is disabled.
exHSRight4H	443	Specifies the forth additional right button (R4) when the cursor hovers it.
exHSRight5	444	Specifies the fifth additional right button (R5) in normal state.
exHSRight5P	445	Specifies the fifth additional right button (R5) when it is pressed.
exHSRight5D	446	Specifies the fifth additional right button (R5) when it is disabled.
exHSRight5H	447	Specifies the fifth additional right button (R5) when the cursor hovers it.

exHSRight6	448	Specifies the sixth additional right button (R6) in normal state.
exHSRight6P	449	Specifies the sixth additional right button (R6) when it is pressed.
exHSRight6D	450	Specifies the sixth additional right button (R6) when it is disabled.
exHSRight6H	451	Specifies the sixth additional right button (R6) when the cursor hovers it.
exSBtn	324	Specifies all button parts (L1-L5, <, exThumbPart, >, R1-R6), in normal state. Use this option the same visual appearance for all buttons in the control.
exSBtnP	325	Specifies all button parts (L1-L5, <, exThumbPart, >, R1-R6), when it is pressed. Use this option the same visual appearance for all buttons in the control.
exSBtnD	326	Specifies all button parts (L1-L5, <, exThumbPart, >, R1-R6), when it is disabled. Use this option the same visual appearance for all buttons in the control.
exSBtnH	327	Specifies all button parts (L1-L5, <, exThumbPart, >, R1-R6), when the cursor hovers it . Use this option the same visual appearance for all buttons in the control.
exScrollHoverAll	500	Enables or disables the hover-all feature. By default (Background(exScrollHoverAll) = 0), the left/top, right/bottom and thumb parts of the control' scrollbars are displayed in hover state while the cursor hovers any part of the scroll bar (hover-all feature). The hover-all feature is available on Windows 11 or greater, if only left/top, right/bottom, thumb, lower and upper-background parts of the scrollbar are visible, no custom visual-appearance is applied to any visible part. The hover-all feature is always on If Background(exScrollHoverAll) = -1. The Background(exScrollHoverAll) = 1 disables the hover-all feature.
exScrollSizeGrip	511	Specifies the background of the control, when the Mode is exSizeGrip.

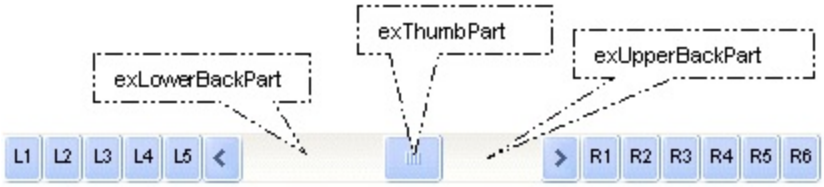
constants ModeEnum

The ModeEnum type specifies the orientation of the control. Use the Mode property to specify the control's orientation. By default, the control's Mode property is exvertical.

Name	Value	Description
exVertical	0	The control is vertically oriented.
exHorizontal	1	The control is horizontally oriented.
exSizeGrip	2	The control displays a size grip.
exThumbProgress	16	The control displays the thumb/scroll box as a progress bar.

constants PartEnum

The PartEnum type defines the parts in the control. Use the VisiblePart or VisibleParts property to specify the parts being shown or hidden in the control. Use the EnablePart or EnableParts property to specify enable or disable parts. Use the OwnerDrawPart property to specify parts that are responsible for its painting. The BtnWidth or BtnHeight property specifies the size of the button. The ThumbSize property specifies the size of the thumb. The parts in the control are explained bellow:



Name	Value	Description
exExtentThumbPart	65536	exExtentThumbPart. The thumb-extension part.
exLeftB1Part	32768	(L1) The first additional button, in the left or top area. By default, this button is hidden.
exLeftB2Part	16384	(L2) The second additional button, in the left or top area. By default, this button is hidden.
exLeftB3Part	8192	(L3) The third additional button, in the left or top area. By default, this button is hidden.
exLeftB4Part	4096	(L4) The forth additional button, in the left or top area. By default, this button is hidden.
exLeftB5Part	2048	(L5) The fifth additional button, in the left or top area. By default, this button is hidden.
exLeftBPart	1024	(<) The left or top button. By default, this button is visible.
exLowerBackPart	512	The area between the left/top button and the thumb. By default, this part is visible.
exThumbPart	256	The thumb part or the scroll box region. By default, the thumb is visible.
exUpperBackPart	128	The area between the thumb and the right/bottom button. By default, this part is visible.
exBackgroundPart	640	The union between the exLowerBackPart and the exUpperBackPart parts. By default, this part is visible.
exRightBPart	64	(>) The right or down button. By default, this button is visible.

exRightB1Part	32	(R1) The first additional button in the right or down side. By default, this button is hidden.
exRightB2Part	16	(R2) The second additional button in the right or down side. By default, this button is hidden.
exRightB3Part	8	(R3) The third additional button in the right or down side. By default, this button is hidden.
exRightB4Part	4	(R4) The forth additional button in the right or down side. By default, this button is hidden.
exRightB5Part	2	(R5) The fifth additional button in the right or down side. By default, this button is hidden.
exRightB6Part	1	(R6) The sixth additional button in the right or down side. By default, this button is hidden.
exPartNone	0	No part.

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 ScrollEnum

The ScrollEnum type specifies the actions that Scroll method can perform.

Name	Value	Description
exScrollLeft	0	Scrolls left/up by one unit. (SmallChange property indicates the unit). Simulates a single click in the control's left/up button.
exScrollRight	1	Scrolls right by one unit. (SmallChange property indicates the unit). Simulates a single click in the control's right/down button.
exScrollPageLeft	2	Scrolls left/up by one page. (LargeChange property indicates the page). Simulates a single click in the control's exLowerBackPart part.
exScrollPageRight	3	Scrolls right/down by one page. (LargeChange property indicates the page). Simulates a single click in the control's exUpperBackPart part.
exScrollToPosition	4	Scrolls the control to specified position. Simulates click the thumb and drags to a new position.

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.
RenderType	Specifies the way colored EBN objects are displayed on the component.

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

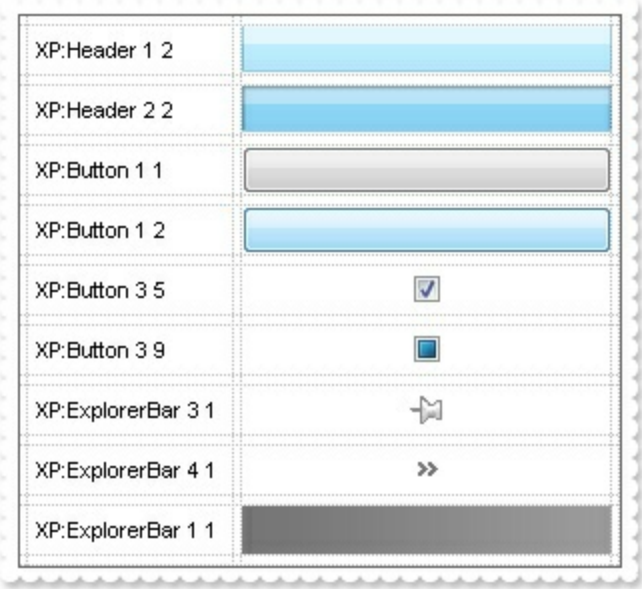
Adds or replaces a skin object to the control.

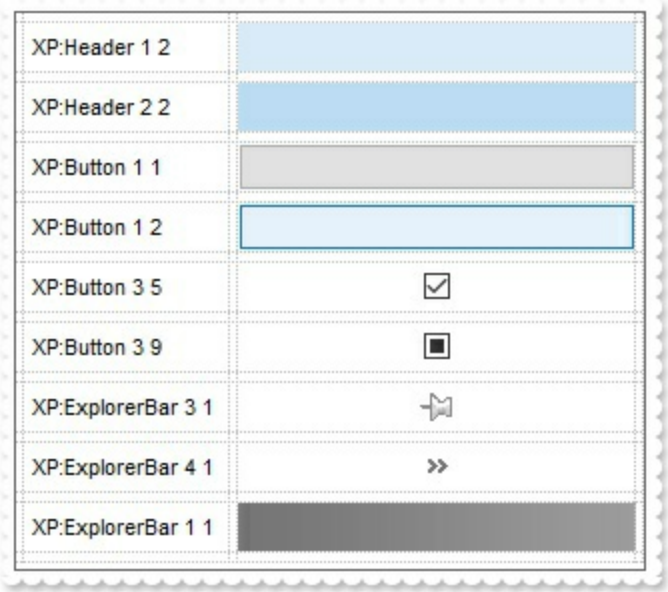
Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
	The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays 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

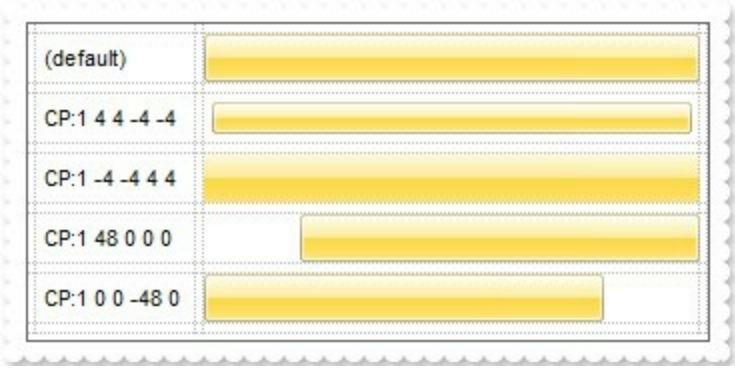




- A copy of another skin with different coordinates (position, size), if the Skin parameter starts with "**CP**:". Use this option, to display the EBN, using different coordinates (position, size). By default, the EBN skin object is rendered on the part's client area. Using this option, you can display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

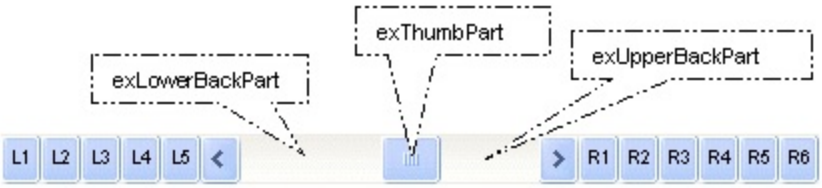
where the ID is the identifier of the EBN to be used (it is a number that specifies the ID parameter of the Add method), Left, Top, Right and Bottom parameters/numbers specifies the relative position to the part's client area, where the EBN should be rendered. The Left, Top, Right and Bottom parameters 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:



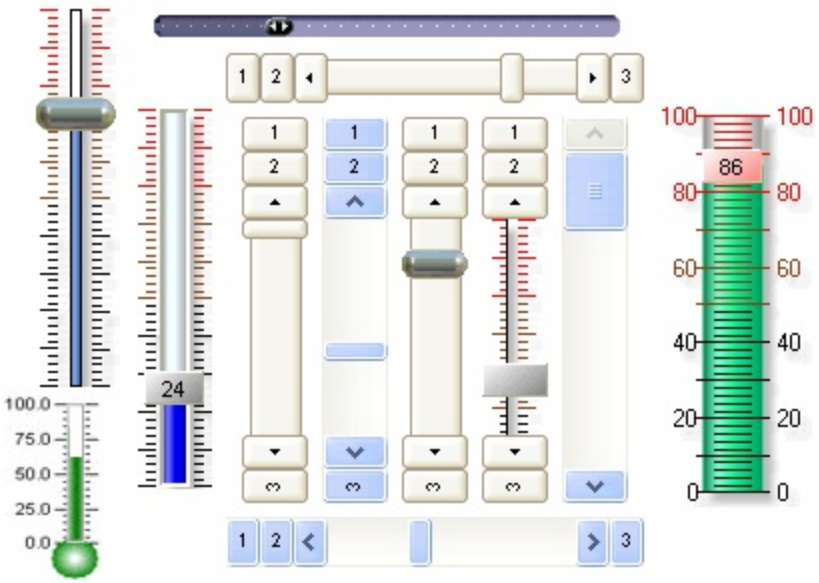
Return	Description
Boolean	A Boolean expression that indicates whether the new skin was added or replaced.

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:

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

```
With ScrollBar1
   .VisualAppearance.Add 1, "D:\Exontrol\ExScrollBar\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 scrollbar:

```
With ScrollBar1
   .VisualAppearance.Add 1, "D:\Exontrol\ExScrollBar\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 scrollbar:

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

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

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

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

```
With AxScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumb, &H1000000)
End With
```

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

```
With AxScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumbP, &H1000000)
End With
```

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

```
axScrollBar1.VisualAppearance.Add(1,
"D:\\Exontrol\\ExScrollBar\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn");
axScrollBar1.set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumb,
0x1000000);
```

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

```
axScrollBar1.VisualAppearance.Add(1,
"D:\\Exontrol\\ExScrollBar\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn");
axScrollBar1.set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumbP,
0x1000000);
```

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

```
with thisform.ScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\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 scrollbar:

```
with thisform.ScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\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.

property Appearance.RenderType as Long

Specifies the way colored EBN objects are displayed on the component.

Type	Description
Long	A long expression that indicates how the EBN objects are shown in the control, like explained bellow.

By default, the RenderType property is 0, which indicates an A-color scheme. The RenderType property can be used to change the colors for the entire control, for parts of the controls that uses EBN objects. The RenderType property is not applied to the currently XP-theme if using.

The RenderType property is applied to all parts that displays an EBN object. The properties of color type may support the EBN object if the property's description includes "*A color expression that indicates the cell's background color. 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.*" In other words, a property that supports EBN objects should be of format 0xIDRRGGBB, where the ID is the identifier of the EBN to be applied, while the BBGGRR is the (Red,Green,Blue, RGB-Color) color to be applied on the selected EBN. For instance, the 0x1000000 indicates displaying the EBN as it is, with no color applied, while the 0x1FF0000, applies the Blue color (RGB(0x0,0x0,0xFF), RGB(0,0,255) on the EBN with the identifier 1. You can use the EBNColor tool to visualize applying EBN colors.

Click here to watch a movie on how you can change the colors to be applied on EBN objects.

For instance, the following sample changes the control's header appearance, by using an EBN object:

```
With Control
   .VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
   .BackColorHeader = &H1000000
End With
```

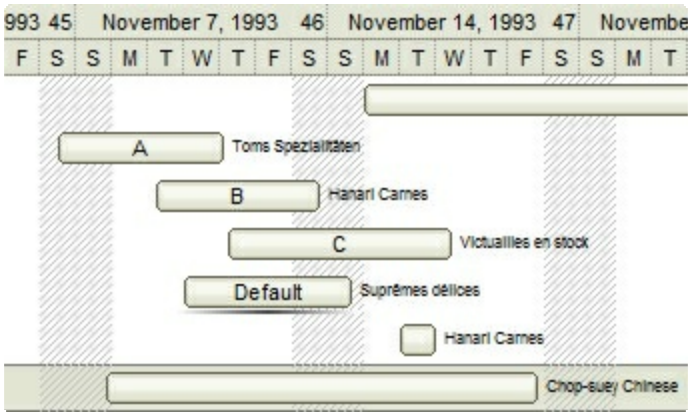
In the following screen shot the following objects displays the current EBN with a different color:

- "A" in Red (RGB(255,0,0), for instance the bar's property exBarColor is 0x10000FF
- "B" in Green (RGB(0,255,0), for instance the bar's property exBarColor is 0x100FF00

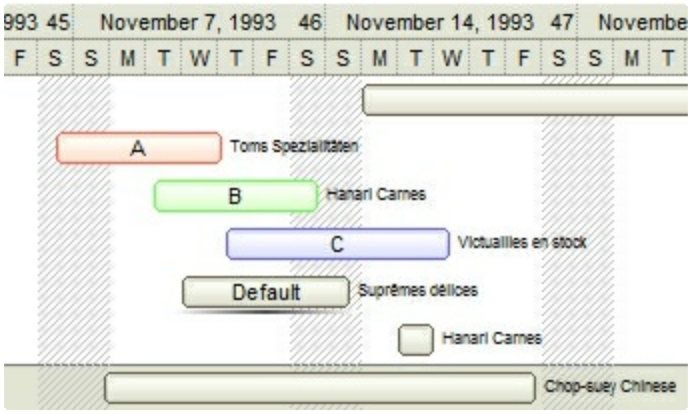
- "C" in Blue (RGB(0,0,255), for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

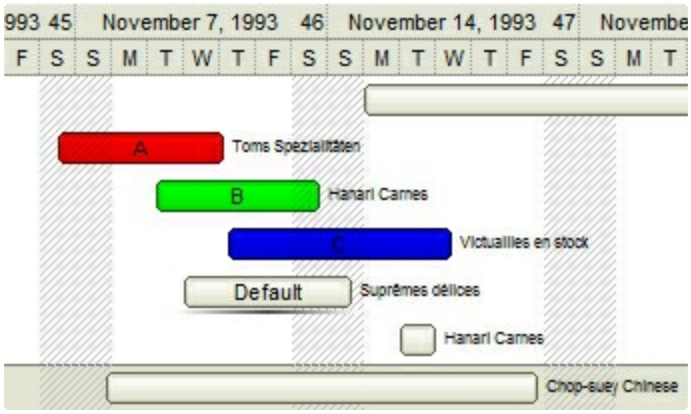
- ***-3**, no color is applied*. For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.



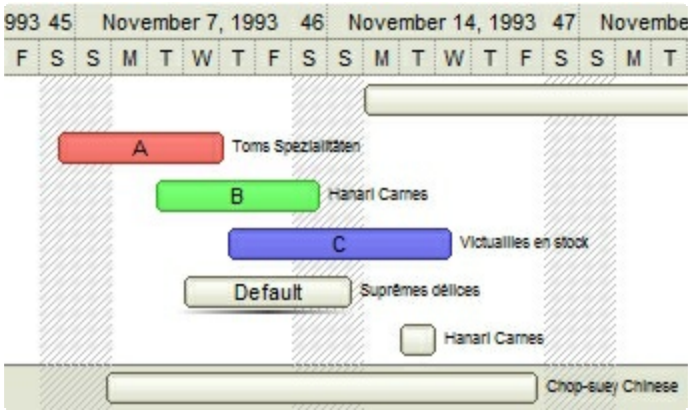
- ***-2**, OR-color scheme*. The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the part of the control. This option should be used with solid colors (RGB(255,0,0), RGB(0,255,0), RGB(0,0,255), RGB(255,255,0), RGB(255,0,255), RGB(0,255,255),RGB(127,0,0),RGB(0,127,0), ...)



- ***-1**, AND-color scheme*, The color to be applied on the part of the control is an AND bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the AND bit for the entire Blue channel, or in other words, it applies a more Blue to the part of the control. This option should be used with solid colors (RGB(255,0,0), RGB(0,255,0), RGB(0,0,255), RGB(255,255,0), RGB(255,0,255), RGB(0,255,255),RGB(127,0,0),RGB(0,127,0), ...)

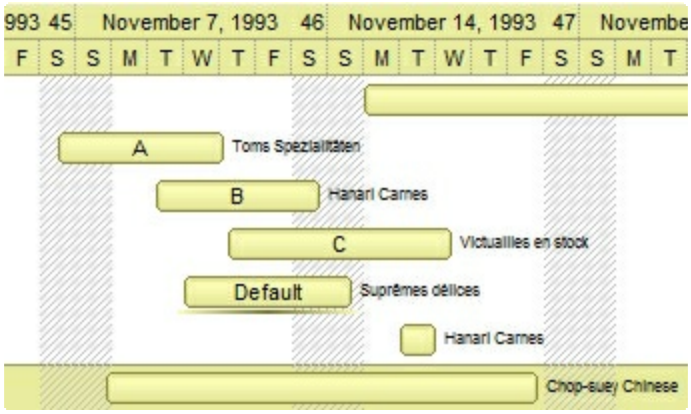


- ***0**, default*, the specified color is applied to the EBN. For instance, the BackColorHeader = &H1FF0000, applies a Blue color to the object. This option could be used to specify any color for the part of the components, that support EBN objects, not only solid colors.

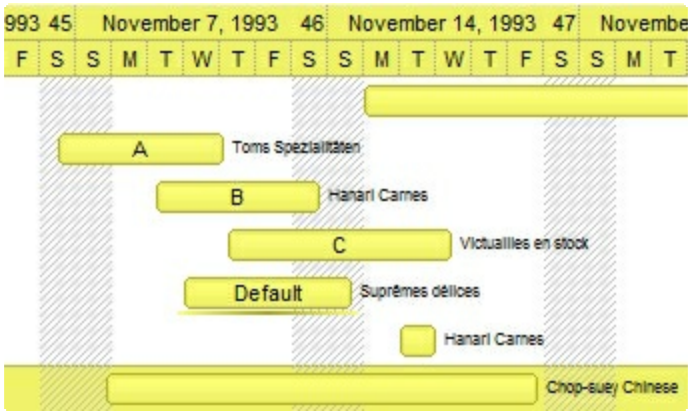


- **0xAABBGGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on 0x4000FFFF, indicates a 25% Yellow on EBN objects. The 0x40, or 64 in decimal, is a 25 % from in a 256 interal, and the 0x00FFFF, indicates the Yellow (RGB(255,255,0)). The same could be if the RenderType is 0x40000000 + vbYellow, or &H40000000 + RGB(255, 255, 0), and so, the RenderType could be the 0xAA000000 + Color, where the Color is the RGB format of the color.

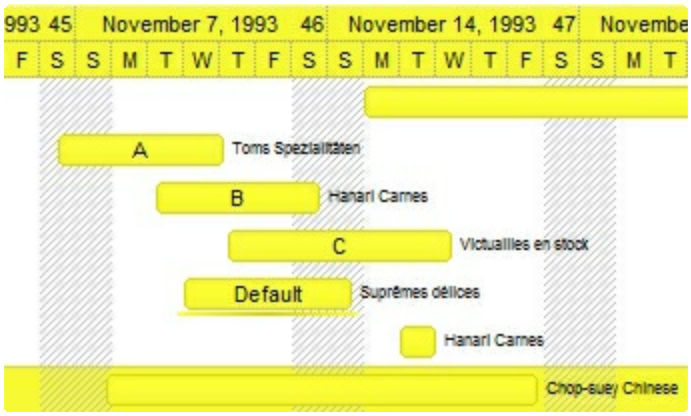
*The following picture shows the control with the RenderType property on 0x**40**00FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256):*



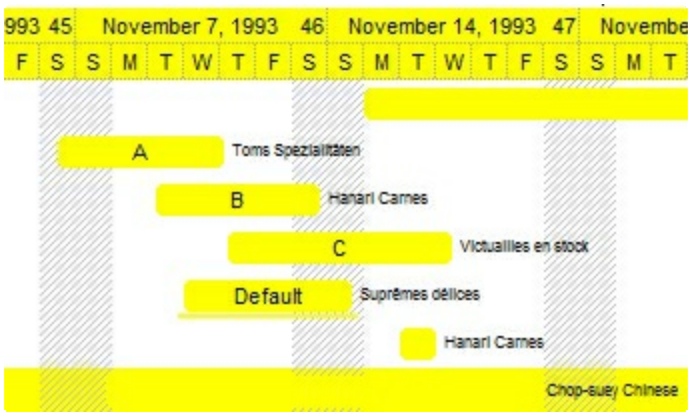
*The following picture shows the control with the RenderType property on 0x**80**00FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256):*



*The following picture shows the control with the RenderType property on 0x**C0**00FFFF (75% Yellow, 0xC0 or 192 in decimal is 75% from 256):*



*The following picture shows the control with the RenderType property on 0x**FF**00FFFF (100% Yellow, 0xFF or 255 in decimal is 100% from 255):*


993 45
November 7, 1993
46
November 14, 1993
47
Novembe
F S S M T W T F S S M T W T F S S M T
A
Toms Spezialitäten
B
Hanari Carnes
C
Victuailles en stock
Default
Suprêmes délices
Hanari Carnes
Chop-suey Chinese

ScrollBar 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: {5DE2B956-5AB5-47EE-8225-6AB7F9B4FC18}. The object's program identifier is: "Exontrol.ScrollBar". The /COM object module is: "ExScrollBar.dll"

The ScrollBar component supports the following properties and methods:

Name	Description
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.
BtnHeight	Specifies the height of the button in the control.
BtnWidth	Specifies the width of the button in the control.
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.
DisableNoScroll	Disables the scroll bar instead of removing it, if the scroll bar's new parameters make the scroll bar unnecessary.
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.
hWndMouseWheel	Associates a window with the current scroll bar when using the mouse wheel over or while it is focused.
IgnoreLargeChange	Ignores the large change value when getting the maximum value.
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..
LargeChange	The amount by which the scroll box position changes when the user clicks in the scroll bar or presses the PAGE UP or PAGE DOWN keys.
Maximum	The upper limit value of the scrollable range.
Minimum	The lower limit value of the scrollable range.
Mode	Specifies the control's Mode.
OrderParts	Specifies the order of the parts in the scroll-bar.
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.
Scroll	Scrolls programmatically the control.
ScrollDelay	Specifies the time in ms, to delay the next scroll event, when the user clicks the scrollbar's parts.
SendMessage	Specifies whether the control sends scroll messages to the parent window.
ShowImageList	Specifies whether the control's image list window is visible or hidden.
SmallChange	The amount by which the scroll box position changes when the user clicks a scroll arrow or presses an arrow key.
StartScrollDelay	Specifies the time in ms, to wait until contiguously scroll begins once the user presses the up/down or left/right

	buttons.
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.
TemplateResult	Gets the result of the last Template call.
TemplateResultN	Gets the result of the last Template call, as double.
TemplateResultS	Gets the result of the last Template call, as string.
ThumbSize	Specifies the width or the height of the thumb.
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 scroll box position represents.
ValueFromPoint	Retrieves the value from the point.
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.
WheelChange	The amount by which the scroll box position changes when the user rolls the mouse wheel.

property ScrollBar.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 buttons and thumb 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 ScrollBar1
   .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 AxScrollBar1
   .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:

```
axScrollBar1.BeginUpdate();
axScrollBar1.VisualAppearance.Add(0x16, "c:\\temp\\frame.ebn");
axScrollBar1.Appearance = (EXSCROLLBARLib.AppearanceEnum)0x16000000;
axScrollBar1.BackColor = Color.FromArgb(250, 250, 250);
axScrollBar1.EndUpdate();
```

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

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

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

```
with thisform.ScrollBar1
    .BeginUpdate
        .VisualAppearance.Add(0x16, "c:\temp\frame.ebn")
        .Appearance = 0x16000000
        .BackColor = RGB(250, 250, 250)
    .EndUpdate
endwith
```

method ScrollBar.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 ScrollBar1_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 ScrollBar.BackColor as Color

Specifies the control's background color.

Type	Description
Color	A Color expression that indicates the

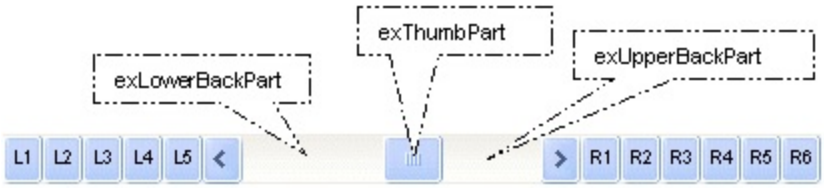
Use the BackColor property to specify the control's background color. This property does not affect the visual appearance of the control applied using the Background property. Use the Picture property to assign a picture on the control's background. Use the ForeColor property to specify the control's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

property ScrollBar.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. Use the OrderParts to specify the order of the buttons in the scroll bar.



In the following samples, we have used the following skin file:

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

```
With ScrollBar1
   .VisualAppearance.Add 1, "D:\Exontrol\ExScrollBar\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 scrollbar:

```
With ScrollBar1
   .VisualAppearance.Add 1, "D:\Exontrol\ExScrollBar\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 scrollbar:

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

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

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

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

```
With AxScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumb, &H1000000)
End With
```

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

```
With AxScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumbP, &H1000000)
End With
```

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

```
axScrollBar1.VisualAppearance.Add(1,
"D:\\Exontrol\\ExScrollBar\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn");
axScrollBar1.set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumb,
0x1000000);
```

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

```
axScrollBar1.VisualAppearance.Add(1,
"D:\\Exontrol\\ExScrollBar\\sample\\VB\\Gauge\\Vertical 2\\thumb.ebn");
axScrollBar1.set_Background(EXSCROLLBARLib.BackgroundPartEnum.exVSThumbP,
0x1000000);
```

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

```
with thisform.ScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\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 scrollbar:

```
with thisform.ScrollBar1
   .VisualAppearance.Add(1, "D:\Exontrol\ExScrollBar\sample\VB\Gauge\Vertical
2\thumb.ebn")
   .Background(261) = 0x1000000
endwith
```

method ScrollBar.BeginUpdate ()

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

Type	Description

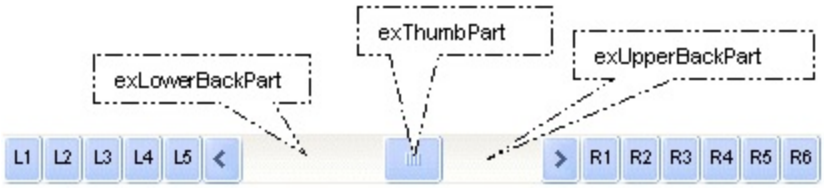
This method prevents the control from painting until the EndUpdate method is called. The BeginUpdate and EndUpdate methods increases the speed of making your changes, by preventing painting the control when it suffers any change. Once that BeginUpdate method was called, you have to make sure that EndUpdate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

property ScrollBar.BtnHeight as Long

Specifies the height of the button in the control.

Type	Description
Long	A long expression that defines the height of the button in a vertical scroll bar.

By default, the BtnHeight property is -1. If the BtnHeight property is -1, the control gets the default's button height from the system. If the BtnHeight property is greater than 0, it indicates in pixels the height of the button in a vertical scroll bar. Use the Mode property to specify whether the control is vertically or horizontally oriented. Use the BtnWidth property to specify the width of the buttons in a horizontal scroll bar.



The BtnHeight property assigns the height for the following buttons:

- L1 to L5
- <
- >
- R1 to R6

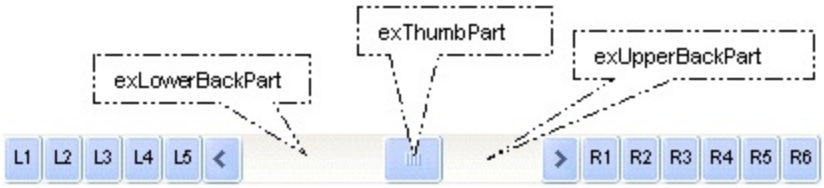
Use the ThumbSize property to define a static size for the control's scrollbox.

property ScrollBar.BtnWidth as Long

Specifies the width of the button in the control.

Type	Description
Long	A long expression that defines the width of the button in a horizontal scroll bar.

By default, the BtnWidth property is -1. If the BtnWidth property is -1, the control gets the default's button width from the system. If the BtnWidth property is greater than 0, it indicates in pixels the width of the button in a horizontal scroll bar. Use the Mode property to specify whether the control is vertically or horizontally oriented. Use the BtnHeight property to specify the width of the buttons in a vertical scroll bar.



The BtnWidth property assigns the width for the following buttons:

- L1 to L5
- <
- >
- R1 to R6

Use the ThumbSize property to define a static size for the control's scrollbox.

property ScrollBar.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 Caption property supports the following built-in HTML tags:

- **bold** bolds a part of the caption.
- <u> underline </u> specifies that the portion should appear as underlined.
- <s> ~~strikeout~~ </s> specifies that the portion should appear as strikeout.
- *italic* specifies that the portion should appear as italic.
- <fgcolor=FF0000>fgcolor</fgcolor> changes the foreground color for a portion.
- <bgcolor=FF0000>bgcolor</bgcolor> changes the background color for a portion.
-
 breaks a line.
- <solidline> draws a solid line. If has no effect for a single line caption.
- <dotline> draws a dotted line. If has no effect for a single line caption.
- <upline> draws the line to the top of the text line
- <r> aligns the rest of the text line to the right side. It has no effect if the caption contains a single line.
- <img>number[:width]</img> inserts an icon inside the cell's caption. The number indicates the index of the icon being inserted. 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 being 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.

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

```
Private Sub ScrollBar1_Change()
   With ScrollBar1
      .Caption(exThumbPart) = .Value
   End With
End Sub
```

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

```
void OnChangeScrollbar1()
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_scrollbar.GetValue() );
   m_scrollbar.SetCaption( 256, strFormat );
}
```

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

```
With AxScrollBar1
   .set_Caption(EXSCROLLBARLib.PartEnum.exThumbPart, .Value.ToString())
End With
```

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

```
private void axScrollBar1_Change(object sender, EventArgs e)
{
   axScrollBar1.set_Caption(EXSCROLLBARLib.PartEnum.exThumbPart,
axScrollBar1.Value.ToString());
}
```

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

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

with thisform.ScrollBar1
   .Caption(256) = .Value
endwith
```

property ScrollBar.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:

- **bold** bolds a part of the caption.
- <u> underline </u> specifies that the portion should appear as underlined.
- <s> ~~strikeout~~ </s> specifies that the portion should appear as strikeout.
- *italic* specifies that the portion should appear as italic.
- <fgcolor=FF0000>fgcolor</fgcolor> changes the foreground color for a portion.
- <bgcolor=FF0000>bgcolor</bgcolor> changes the background color for a portion.
-
 breaks a line.
- <solidline> draws a solid line. If has no effect for a single line caption.
- <dotline> draws a dotted line. If has no effect for a single line caption.
- <upline> draws the line to the top of the text line
- <r> aligns the rest of the text line to the right side. It has no effect if the caption contains a single line.
- <img>number[:width]</img> inserts an icon inside the cell's caption. The number indicates the index of the icon being inserted. 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 being 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.

property ScrollBar.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 ScrollBar.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 ScrollBar.DisableNoScroll as Boolean

Disables the scroll bar instead of removing it, if the scroll bar's new parameters make the scroll bar unnecessary.

Type	Description
Boolean	A Boolean expression that indicates whether DisableNoScroll feature is off or on.

By default, the DisableNoScroll property is False. If the DisableNoScroll property is True, the left/up or right/down buttons are enabled only if they required, else they are disabled. For instance, if the Value property is equal with the Minimum property, the left/up button is disabled. Also, if both buttons are not required, the thumb part is not shown. Use the EnablePart property to enable or disable a specified part. Use the VisiblePart property to specify which parts are visible or hidden. Use the Background property to change the visual appearance for a button or a part in a specified state.

property ScrollBar.Enabled as Boolean

Enables or disables the control.

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

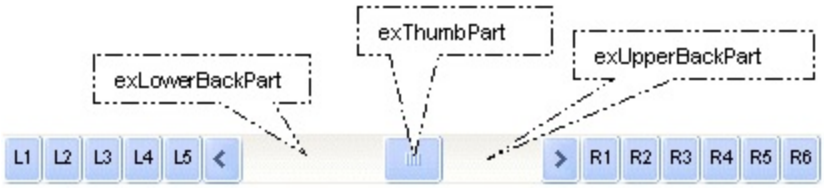
By default, the Enabled property is True. Use the EnablePart or EnableParts property to specify a disabled part. If the Enabled property is False, all visible parts of the control are displayed in disabled state. Use the VisiblePart property to specify which parts are visible or hidden. Use the Background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

property ScrollBar.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. Use the OrderParts to specify the order of the buttons in the scroll bar.



By default, the following parts are enabled:

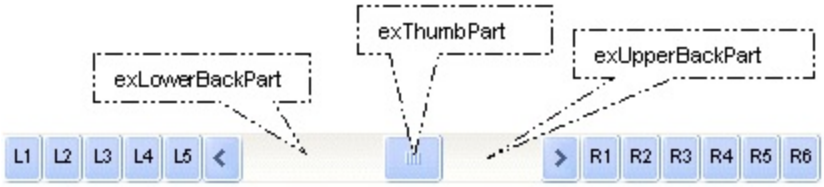
- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

property ScrollBar.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 1984 (that's a OR combination of exLeftBPart, exLowerBackPart, exThumbPart, exUpperBackPart and exRightBPart). 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. Use the OrderParts to specify the order of the buttons in the scroll bar.



By default, the following parts are enabled:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

method ScrollBar.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 ScrollBar.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 ScrollBar.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 String 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). The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

For instance, the following sample retrieves the beginning date (as string) for the default bar in the first visible item:

```
Debug.Print ScrollBar1.ExecuteTemplate("Items.ItemBar(FirstVisibleItem(),``,1)")
```

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 ScrollBar.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 ScrollBar.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 ScrollBar.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 ScrollBar.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.

property ScrollBar.hWndMouseWheel as Long

Associates a window with the current scroll bar when using the mouse wheel over or while it is focused.

Type	Description
Long	A Long expression that specifies the handle of the window to be associated with the current scroll bar. If -1, the mouse wheel works anywhere on the form, or the scroll bar's value is changed every time the user rolls the mouse wheel, no matter what field is focused.

By default, the hWndMouseWheel property is 0, which indicates no window associated with the scroll bar. If non-zero, this property indicates that handle of the window to be associated with the current scroll bar, so while this window has the focus, rotating the mouse wheel will be forward to the scroll bar to change the current value. For instance, let's say we have a text-box and we want to be able to change the value inside while the user rotates the mouse wheel.

property ScrollBar.IgnoreLargeChange as Boolean

Ignores the large change value when getting the maximum value.

Type	Description
Boolean	A boolen expression that indicates the whether the LargeChange property counts in getting the maximum value.

By default, the IgnoreLargeChange property is False. The LargeChange property gets or sets a value to be added to or subtracted from the Value property when the scroll box 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, if the IgnoreLargeChange property is True, or the LargeChange property is 0.
- Minimum to ((Maximum - LargeChange) + 1), if the IgnoreLargeChange property is False and the LargeChange property is not 0.

method ScrollBar.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 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 ScrollBar.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 ScrollBar.LargeChange as Long

The amount by which the scroll box position changes when the user clicks in the scroll bar 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 scroll's box 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 scroll box 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 Value property goes from:

- Minimum to Maximum values, if the IgnoreLargeChange property is True, or the LargeChange property is 0.
- Minimum to ((Maximum - LargeChange) + 1), if the IgnoreLargeChange property is False and the LargeChange property is not 0.

property ScrollBar.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 100. 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 scroll box 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, if the IgnoreLargeChange property is True, or the LargeChange property is 0.
- Minimum to ((Maximum - LargeChange) + 1), if the IgnoreLargeChange property is False and the LargeChange property is not 0.

property ScrollBar.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 scroll box 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, if the IgnoreLargeChange property is True, or the LargeChange property is 0.
- Minimum to ((Maximum - LargeChange) + 1), if the IgnoreLargeChange property is False and the LargeChange property is not 0.

property ScrollBar.Mode as ModeEnum

Specifies the control's Mode.

Type	Description
ModeEnum	A ModeEnum expression that indicates the control's orientation.

By default, the Mode property is exVertical. Use the Mode property to change the control's orientation. Use the Value property to specify the control's value. Use the SendMessage property to specify whether the control sends scroll messages to the parent window. If the SendMessage property is True, the control sends scroll bar related messages to the parent window of the control, when certain actions occurs in the control. For instance, if the user clicks the left button in an horizontal message, the control sends the WM_HSCROLL message with the code SB_LINELEFT. Use the SendMessage property only when you need to handle messages in your parent window.

property ScrollBar.OrderParts as String

Specifies the order of the parts in the scroll-bar.

Type	Description
String	A String expression that indicates the order of the parts. The list includes expressions like l, l1, ..., l5, t, r, r1, ..., r6 separated by comma, each expression indicating a part of the control, and its position indicating the displaying order.

Use the OrderParts to customize the order of the buttons in the scroll bar. By default, the OrderParts property is empty. If the OrderParts property is empty the default order of the parts in the control are displayed like follows:

so, the order of the parts is: l1, l2, l3, l4, l5, l, t, r, r1, r2, r3, r4, r5 and r6. Use the VisiblePart or VisibleParts property to specify whether a part in the scrollbar is visible or hidden. Use the EnablePart property to enable or disable a part in the scroll bar. Use the Caption property to assign a caption to a button in the scroll bar.

Use the OrderParts property to change the order of the parts in the control. For instance, "l,r,t,l1,r1" puts the left and right buttons to the left of the thumb area, and the l1 and r1 buttons right after the thumb area. If the parts are not specified in the OrderParts property, automatically they are added to the end.

The list of supported literals in the OrderParts property is:

- **l** for exLeftBPart, (<) The left or top button.
- **l1** for exLeftB1Part, (L1) The first additional button, in the left or top area.
- **l2** for exLeftB2Part, (L2) The second additional button, in the left or top area.
- **l3** for exLeftB3Part, (L3) The third additional button, in the left or top area.
- **l4** for exLeftB4Part, (L4) The forth additional button, in the left or top area.
- **l5** for exLeftB5Part, (L5) The fifth additional button, in the left or top area.
- **t** for exLowerBackPart, exThumbPart and exUpperBackPart, The union between the exLowerBackPart and the exUpperBackPart parts.
- **r** for exRightBPart, (>) The right or down button.
- **r1** for exRightB1Part, (R1) The first additional button in the right or down side.
- **r2** for exRightB2Part, (R2) The second additional button in the right or down side.
- **r3** for exRightB3Part, (R3) The third additional button in the right or down side.
- **r4** for exRightB4Part, (R4) The forth additional button in the right or down side.

- **r5** for exRightB5Part, (R5) The fifth additional button in the right or down side.
- **r6** for exRightB6Part, (R6) The sixth additional button in the right or down side.

Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property ScrollBar.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. For instance, the VisiblePart(exLeftB1Part or exLeftB2Part) = True adds two new buttons left/up to the control.

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

- exBackgroundPart
- exLowerBackPart
- exUpperBackPart
- exLeftBPart.
- exLeftB1Part
- exLeftB2Part
- exLeftB3Part
- exLeftB4Part
- exLeftB5Part
- exRightBPart
- exRightB1Part
- exRightB2Part
- exRightB3Part
- exRightB4Part
- exRightB5Part
- exRightB6Part

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

```
With ScrollBar1
   .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 ScrollBar1_OwnerDrawEnd(ByVal Part As EXSCROLLBARLibCtl.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_scrollbar.SetOwnerDrawPart( 128 /*exUpperBackPart*/,  TRUE );
m_scrollbar.SetOwnerDrawPart( 512 /*exLowerBackPart*/,  TRUE );
```

```
void OnOwnerDrawEndScrollbar1(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 ScrollBar.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 ScrollBar1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With ScrollBar1
      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 AxScrollBar1_MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent) Handles
AxScrollBar1.MouseDownEvent
   With AxScrollBar1
      If (0 <> (.get_PartFromPoint(-1, -1) And
EXSCROLLBARLib.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 OnMouseDownScrollbar1(short Button, short Shift, long X, long Y)
{
  if ( m_scrollbar.GetPartFromPoint(-1,-1) & 640 )
    m_scrollbar.SetValue( m_scrollbar.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 axScrollBar1_MouseDownEvent(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent e)
{
  if ( 0 != ( axScrollBar1.get_PartFromPoint(-1,-1) &
EXSCROLLBARLib.PartEnum.exBackgroundPart ) )
    axScrollBar1.Value = axScrollBar1.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.scrollbar1
  if ( 0 # bitand( .PartFromPoint(-1,-1), 640) )
    .Value = .ValueFromPoint(-1,-1)
  endif
endwith
```

property ScrollBar.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 ScrollBar.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 ScrollBar.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. The ImageSize property defines the size (width/height) of the icons within 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.

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

i = ExScrollBar1.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 = ExScrollBar1.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:

ExScrollBar1.ReplaceIcon 0, i, i specifies the index of icon removed.

The following VB clears the control's icons collection:

ExScrollBar1.ReplaceIcon 0, -1

method ScrollBar.Scroll (Action as ScrollEnum, ToPosition as Long)

Scrolls programmatically the control.

Type	Description
Action as ScrollEnum	A ScrollEnum expression that specifies the action to perform.
ToPosition as Long	A long expression that indicates the new position, when the Action is is exScrollToPosition

use the Scroll method to simulate actions when some parts are clicked. For instance, Scroll(exScrollLeft) simulates a single click in the control's left/up button, so the Value property is decreased with the SmallChange value. The Scroll method does not fire events like ClickPart or ClickingPart. Use the Value property to specify the control's value.

property ScrollBar.ScrollDelay as Long

Specifies the time in ms, to delay the next scroll event, when the user clicks the scrollbar's parts.

Type	Description
Long	A Long expression that defines the time in ms, to delay the next scroll event.

By default, the ScrollDelay property is 0, which means that no delay occurs till next scroll event. In other words, you can use the ScrollDelay property to define the delay between two Change events. Use the StartScrollDelay property to specify the time in ms, to wait until contiguously scroll begins once the user presses the up/down or left/right buttons.

property ScrollBar.SendMessage as Boolean

Specifies whether the control sends scroll messages to the parent window.

Type	Description
Boolean	A Boolean expression that specifies whether the control sends scroll bar related messages to the parent window of the control.

Use the SendMessage property to specify whether the control sends scroll messages to the parent window. If the SendMessage property is True, the control sends scroll bar related messages to the parent window of the control, when certain actions occurs in the control. For instance, if the user clicks the left button in an horizontal message, the control sends the WM_HSCROLL message with the code SB_LINELEFT. Use the SendMessage property only when you need to handle messages in your parent window. Use the Mode property to specify the control's orientation.

property ScrollBar.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 ScrollBar 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 ScrollBar.SmallChange as Long

The amount by which the scroll box position changes when the user clicks a scroll arrow or presses an arrow key.

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 scroll box 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.

property ScrollBar.StartScrollDelay as Long

Specifies the time in ms, to wait until contiguously scroll begins once the user presses the up/down or left/right buttons.

Type	Description
Long	A Long expression that defines the time in ms, to wait until contiguously scroll begins once the user presses the up/down or left/right buttons.

By default, the StartScrollDelay property is 500, which means that the scrolling begins after 1/2 seconds once the user clicks any up/down or left/right buttons. Use the ScrollDelay property to define the delay in ms, between two Change events.

property ScrollBar.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). The TemplateResult property returns the result of the last Template call. Use the ExecuteTemplate property to execute a template script and gets the result. The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

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 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 ScrollBar.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 ScrollBar.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 ScrollBar.TemplateResult as Variant

Gets the result of the last Template call.

Type	Description
Variant	A VARIANT expression that indicates the result of the last Template call. The TemplateResultN property gets the result as number (double expression). The TemplateResultS property gets the result as string.

The TemplateResult, TemplateResultN, TemplateResultS property returns the result of the last Template call, as variant, numeric (double) or as string. The Template property takes a string called x-script, and executes it. For instance, you can use the TemplateDef, Template, TemplateResult or ExecuteTemplate to work with x-script. It is known that programming languages such as **dBASE Plus, XBasic from AlphaFive, Wonderware**, 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.

For instance, the Wonderware does not support parameters for events, or parameters of any event are not defined during the event, so in this case, you require an alternative in order to get the value for these parameters. Let's say the Select event, which has one parameter ID of long type, which indicates the identifier of the item being selected. The EventParam property gets the value for any parameter of a specified event. The same, the EventParam requires parameters so Wonderware won't support it, in this case, the Template and TemplateResult can be used to get the ID parameter of the Select event as follows:

```
DIM id As Message
#exMenu1.Template = "EventParam(0)";
id = #exMenu1.TemplateResultS;
MessageBox(id, "Identifier", 0);
```

This code must be called during the Select event, else the EventParam has no effect.

The Template script (x-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 ScrollBar.TemplateResultN as Double

Gets the result of the last Template call, as double.

Type	Description
Double	A Double expression that indicates the result of the last Template call. The TemplateResult property gets the result as variant. The TemplateResultS property gets the result as string.

The TemplateResult, TemplateResultN, TemplateResultS property returns the result of the last Template call, as variant, numeric (double) or as string. The Template property takes a string called x-script, and executes it. For instance, you can use the TemplateDef, Template, TemplateResult or ExecuteTemplate to work with x-script. It is known that programming languages such as **dBASE Plus, XBasic from AlphaFive, Wonderware**, 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.

For instance, the Wonderware does not support parameters for events, or parameters of any event are not defined during the event, so in this case, you require an alternative in order to get the value for these parameters. Let's say the Select event, which has one parameter ID of long type, which indicates the identifier of the item being selected. The EventParam property gets the value for any parameter of a specified event. The same, the EventParam requires parameters so Wonderware won't support it, in this case, the Template and TemplateResult can be used to get the ID parameter of the Select event as follows:

```
DIM id As Message
#exMenu1.Template = "EventParam(0)";
id = #exMenu1.TemplateResultS;
MessageBox(id, "Identifier", 0);
```

This code must be called during the Select event, else the EventParam has no effect.

The Template script (x-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 ScrollBar.TemplateResultS as String

Gets the result of the last Template call, as string.

Type	Description
String	A String expression that indicates the result of the last Template call. The TemplateResultN property gets the result as number (double expression). The TemplateResult property gets the result as variant.

The TemplateResult, TemplateResultN, TemplateResultS property returns the result of the last Template call, as variant, numeric (double) or as string. The Template property takes a string called x-script, and executes it. For instance, you can use the TemplateDef, Template, TemplateResult or ExecuteTemplate to work with x-script. It is known that programming languages such as **dBASE Plus, XBasic from AlphaFive, Wonderware**, 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.

For instance, the Wonderware does not support parameters for events, or parameters of any event are not defined during the event, so in this case, you require an alternative in order to get the value for these parameters. Let's say the Select event, which has one parameter ID of long type, which indicates the identifier of the item being selected. The EventParam property gets the value for any parameter of a specified event. The same, the EventParam requires parameters so Wonderware won't support it, in this case, the Template and TemplateResult can be used to get the ID parameter of the Select event as follows:

```
DIM id As Message
#exMenu1.Template = "EventParam(0)";
id = #exMenu1.TemplateResultS;
MessageBox(id, "Identifier", 0);
```

This code must be called during the Select event, else the EventParam has no effect.

The Template script (x-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 ScrollBar.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 Mode property to specify whether the control is vertically or horizontally oriented. Use the BtnHeight property to specify the width of the buttons in a vertical scroll bar. Use the BtnWidth property to specify the width of the buttons in a horizontal scroll bar.

You can use the ThumbSize property on 0, to allow the control acts like a spin control. The Change event occurs when the control's Value property is changed, or the user clicks the up/down, left/right buttons. The ClickPart(Part) event notifies once the user clicks a part of the control. The ClickingPart(Part) event is fired continuously while the user keeps clicking the part of the control.

property ScrollBar.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 ScrollBar.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 ScrollBar1_Change()
   With ScrollBar1
      .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 AxScrollBar1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxScrollBar1.Change
   With AxScrollBar1
      .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 axScrollBar1_Change(object sender, EventArgs e)
{
   axScrollBar1.ToolTipText = "Record " + axScrollBar1.Value.ToString() + "/" +
axScrollBar1.Maximum.ToString();
```

```
    axScrollBar1.ToolTipTitle = "Position";
}
```

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

```
void OnChangeScrollbar1()
{
    CString strFormat;
    strFormat.Format( _T("Record %i/%i"), m_scrollbar.GetValue(),
m_scrollbar.GetMaximum() );
    m_scrollbar.SetToolTipText( strFormat );
    m_scrollbar.SetToolTipTitle( "Position" );
}
```

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

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

with thisform.ScrollBar1
    .Object.ToolTipText = "Record " + ltrim(str(.Value)) + "/" + ltrim(str(.Maximum))
    .ToolTipTitle = "Position"
endwith
```

property ScrollBar.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 ScrollBar1_Change()
   With ScrollBar1
      .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 AxScrollBar1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxScrollBar1.Change
   With AxScrollBar1
      .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 axScrollBar1_Change(object sender, EventArgs e)
{
   axScrollBar1.ToolTipText = "Record " + axScrollBar1.Value.ToString() + "/" +
axScrollBar1.Maximum.ToString();
```

```
    axScrollBar1.ToolTipTitle = "Position";
}
```

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

```
void OnChangeScrollbar1()
{
    CString strFormat;
    strFormat.Format( _T("Record %i/%i"), m_scrollbar.GetValue(),
m_scrollbar.GetMaximum() );
    m_scrollbar.SetToolTipText( strFormat );
    m_scrollbar.SetToolTipTitle( "Position" );
}
```

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

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

with thisform.ScrollBar1
    .Object.ToolTipText = "Record " + ltrim(str(.Value)) + "/" + ltrim(str(.Maximum))
    .ToolTipTitle = "Position"
endwith
```

property ScrollBar.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 ScrollBar.ToolTipX as String

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

Type	Description
String	A String expression that defines the position (x-position is screen coordinate), where the tooltip will be shown.

By default, the ToolTipX property is empty, which indicates that the tooltip is shown at its default position. The ToolTipX property indicates an expression that determines the horizontal-position of the tooltip, in screen coordinates. The expression supports predefined keys such as value, x, y, width and height among other general constants, operators and functions. The value indicates the current horizontal-position the tooltip is shown, in screen coordinates. The x, y indicates the horizontal/vertical position the tooltip is shown expressed in screen coordinates. The width an height keywords specifies the size of the tooltip is about to be shown.

For instance:

- ToolTipX = "value + 16", indicates that the tooltip should be shown 16-pixels right to the default position
- ToolTipX = "128" specifies a fixed horizontal-position for the tooltip, no matter where the thumb is dragged
- ToolTipX = "1024 - width" shows the tooltip left to 1024 x-coordinate.

The ToolTipX property supports the following keywords:

- **value** keyword, specifies the x-position is screen coordinate, where the tooltip is currently displayed
- **x** keyword, specifies the x-position is screen coordinate, where the tooltip is currently displayed
- **y** keyword, specifies the y-position is screen coordinate, where the tooltip is currently displayed
- **width** keyword, specifies the width in pixels, the current tooltip is shown.
- **height** keyword, specifies the height in pixels, the current tooltip is shown.

Also, this property supports all constants, operators and functions defined here.

property ScrollBar.ToolTipY as String

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

Type	Description
String	A String expression that defines the position (x-position is screen coordinate), where the tooltip will be shown.

By default, the ToolTipY property is empty, which indicates that the tooltip is shown at its default position. ToolTipY property indicates an expression that determines the vertical-position of the tooltip, in screen coordinates. The expression supports predefined keys such as value, x, y, width and height among other general constants, operators and functions. The value indicates the current vertical-position the tooltip is shown, in screen coordinates. The x, y indicates the horizontal/vertical position the tooltip is shown expressed in screen coordinates. The width an height keywords specifies the size of the tooltip is about to be shown.

For instance:

- ToolTipY = "value + 16", indicates that the tooltip should be shown 16-pixels down to the default position
- ToolTipY = "128" specifies a fixed vertical-position for the tooltip, no matter where the thumb is dragged
- ToolTipY = "1024 - height" shows the tooltip up to the 1024 y-coordinate.

The ToolTipX property supports the following keywords:

- **value** keyword, specifies the y-position is screen coordinate, where the tooltip is currently displayed
- **x** keyword, specifies the x-position is screen coordinate, where the tooltip is currently displayed
- **y** keyword, specifies the y-position is screen coordinate, where the tooltip is currently displayed
- **width** keyword, specifies the width in pixels, the current tooltip is shown.
- **height** keyword, specifies the height in pixels, the current tooltip is shown.

Also, this property supports all constants, operators and functions defined here.

property ScrollBar.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 ScrollBar.Value as Long

The value that the scroll box position represents.

Type	Description
Long	A long expression that indicates the control's 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 scroll box is moved a large distance. Use the Background property to change the visual appearance for any part of the control, in any state. . The WheelChange property indicates the amount by which the scroll box position changes when the user rolls the mouse wheel.

The Value property goes from:

- Minimum to Maximum values, if the IgnoreLargeChange property is True, or the LargeChange property is 0.
- Minimum to ((Maximum - LargeChange) + 1), if the IgnoreLargeChange property is False and the LargeChange property is not 0.

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

```
Private Sub ScrollBar1_Change()
   With ScrollBar1
      .Caption(exThumbPart) = .Value
   End With
End Sub
```

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

```
void OnChangeScrollbar1()
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_scrollbar.GetValue() );
   m_scrollbar.SetCaption( 256, strFormat );
}
```

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

```
With AxScrollBar1
   .set_Caption(EXSCROLLBARLib.PartEnum.exThumbPart, .Value.ToString())
End With
```

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

```
private void axScrollBar1_Change(object sender, EventArgs e)
{
   axScrollBar1.set_Caption(EXSCROLLBARLib.PartEnum.exThumbPart,
axScrollBar1.Value.ToString());
}
```

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

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

with thisform.ScrollBar1
   .Caption(256) = .Value
endwith
```

property ScrollBar.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.

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 ScrollBar1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   ScrollBar1.Value = ScrollBar1.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 AxScrollBar1_MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent) Handles AxScrollBar1.MouseDownEvent
   AxScrollBar1.Value = AxScrollBar1.get_ValueFromPoint(-1, -1)
End Sub
```

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

```
void OnMouseDownScrollbar1(short Button, short Shift, long X, long Y)
{
   m_scrollbar.SetValue( m_scrollbar.GetValueFromPoint(-1,-1) );
}
```

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

```
private void axScrollBar1_MouseDownEvent(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent e)
{
    axScrollBar1.Value = axScrollBar1.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.scrollbar1
    .Value = .ValueFromPoint(-1,-1)
endwith
```

property ScrollBar.Version as String

Retrieves the control's version.

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

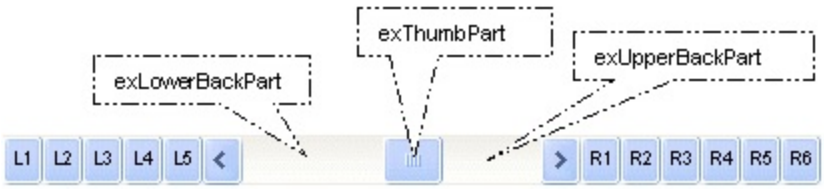
The version property specifies the control's version.

property ScrollBar.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. Use the OrderParts to specify the order of the buttons in the scroll bar.



By default, the following parts are shown:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

For instance the following VB sample adds two additional buttons to the left/up side of the control:

```
With ScrollBar1
   .VisiblePart(exLeftB1Part Or exLeftB2Part) = True
End With
```

The following VB sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub ScrollBar1_ClickPart(ByVal Part As EXSCROLLBARLibCtl.PartEnum)
   If (Part = exLeftB1Part) Then
      MsgBox ("Click")
   End If
End Sub
```

For instance the following VB.NET sample adds two additional buttons to the left/up side of the control:

```
With AxScrollBar1
   .set_VisiblePart(EXSCROLLBARLib.PartEnum.exLeftB1Part Or
EXSCROLLBARLib.PartEnum.exLeftB2Part, True)
End With
```

The following VB.NET sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub AxScrollBar1_ClickPart(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent) Handles AxScrollBar1.ClickPart
   If (e.part = EXSCROLLBARLib.PartEnum.exLeftB1Part) Then
      MsgBox("Click")
   End If
End Sub
```

For instance the following C++ sample adds two additional buttons to the left/up side of the control:

```
m_scrollbar.SetVisiblePart( 32768 /*exLeftB1Part*/ | 16384 /*exLeftB2Part*/, TRUE );
```

The following C++ sample displays a message when the user clicks the exLeftB1Part part of the control:

```
void OnClickPartScrollbar1(long Part)
{
   if ( Part == 32768 /*exLeftB1Part*/ )
      MessageBox( "Click" );
}
```

For instance the following C# sample adds two additional buttons to the left/up side of the control:

```
axScrollBar1.set_VisiblePart(EXSCROLLBARLib.PartEnum.exLeftB1Part |
EXSCROLLBARLib.PartEnum.exLeftB2Part, true);
```

The following C# sample displays a message when the user clicks the exLeftB1Part part of the control:

```
private void axScrollBar1_ClickPart(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent e)
{
   if (e.part == EXSCROLLBARLib.PartEnum.exLeftB1Part)
      MessageBox.Show("Click");
}
```

For instance the following VFP sample adds two additional buttons to the left/up side of the control:

```
with thisform.ScrollBar1
   .VisiblePart(bitor(32768,16384)) = .t.
endwith
```

The following VFP sample displays a message when the user clicks the exLeftB1Part part of the control:

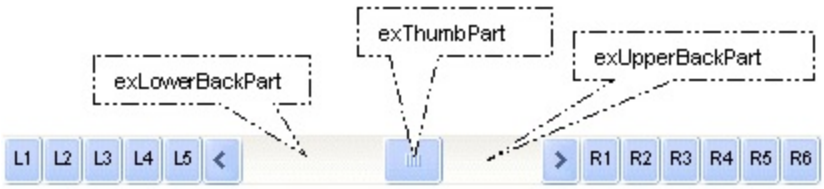
```
*** ActiveX Control Event ***
LPARAMETERS part

if ( part = 32768 )
   wait window "click"
endif
```

property ScrollBar.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 1984 (that's a OR combination of exLeftBPart, exLowerBackPart, exThumbPart, exUpperBackPart and exRightBPart). 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. Use the OrderParts to specify the order of the buttons in the scroll bar.



By default, the following parts are shown:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button 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.

The following VB sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub ScrollBar1_ClickPart(ByVal Part As EXSCROLLBARLibCtl.PartEnum)
  If (Part = exLeftB1Part) Then
    MsgBox ("Click")
  End If
End Sub
```

The following VB.NET sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub AxScrollBar1_ClickPart(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent) Handles AxScrollBar1.ClickPart
   If (e.part = EXSCROLLBARLib.PartEnum.exLeftB1Part) Then
      MsgBox("Click")
   End If
End Sub
```

The following C++ sample displays a message when the user clicks the exLeftB1Part part of the control:

```
void OnClickPartScrollbar1(long Part)
{
   if ( Part == 32768 /*exLeftB1Part*/ )
      MessageBox( "Click" );
}
```

The following C# sample displays a message when the user clicks the exLeftB1Part part of the control:

```
private void axScrollBar1_ClickPart(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent e)
{
   if (e.part == EXSCROLLBARLib.PartEnum.exLeftB1Part)
      MessageBox.Show("Click");
}
```

The following VFP sample displays a message when the user clicks the exLeftB1Part part of the control:

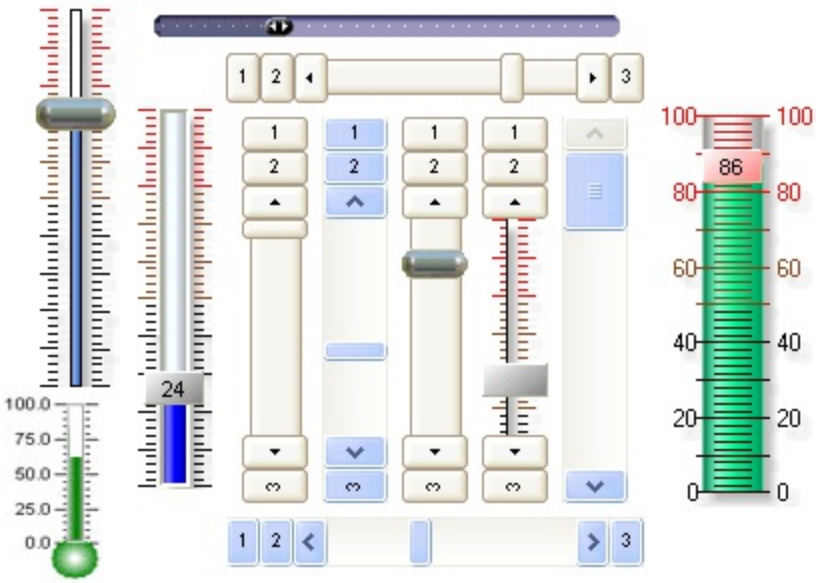
```
*** ActiveX Control Event ***
LPARAMETERS part

if ( part = 32768 )
   wait window "click"
endif
```

property ScrollBar.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.



property ScrollBar.WheelChange as Long

The amount by which the scroll box position changes when the user rolls the mouse wheel.

Type	Description
Long	A Long expression that indicates the amount by which the scroll box position changes when the user rolls the mouse wheel.

By default, the WheelChange property is 1. The WheelChange property indicates the amount by which the scroll box position changes when the user rolls the mouse wheel. The Value property specifies the current scroll position/value.

ExScrollBar 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: {5DE2B956-5AB5-47EE-8225-6AB7F9B4FC18}. The object's program identifier is: "Exontrol.ScrollBar". The /COM object module is: "ExScrollBar.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.
Event	Notifies the application once the control fires an event.
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 is changed. The Value property of the control specifies the value of the control. Use the Minimum and Maximum 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(oScrollBar)
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 ScrollBar1_Change()
   With ScrollBar1
      .Caption(exThumbPart) = .Value
   End With
End Sub
```

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

```
void OnChangeScrollbar1()
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_scrollbar.GetValue() );
   m_scrollbar.SetCaption( 256, strFormat );
}
```

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

```
With AxScrollBar1
   .set_Caption(EXSCROLLBARLib.PartEnum.exThumbPart, .Value.ToString())
End With
```

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

```
private void axScrollBar1_Change(object sender, EventArgs e)
{
    axScrollBar1.set_Caption(EXSCROLLBARLib.PartEnum.exThumbPart,
axScrollBar1.Value.ToString());
}
```

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

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

with thisform.ScrollBar1
    .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 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 scroll box 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,
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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 samples prints the old and the new value on the thumb part of the control:

```
Private Sub ScrollBar1_Changing(ByVal OldValue As Long, NewValue As Long)
   With ScrollBar1
      .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 AxScrollBar1_Changing(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_ChangingEvent) Handles AxScrollBar1.Changing
   With AxScrollBar1
      .set_Caption(EXSCROLLBARLib.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 OnChangingScrollbar1(long OldValue, long FAR* NewValue)
{
   CString strFormat;
   strFormat.Format( _T("<img></img>%i - %i"), OldValue, *NewValue );
   m_scrollbar.SetCaption( 256, strFormat );
}
```

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

```
private void axScrollBar1_Changing(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ChangingEvent e)
{
   axScrollBar1.set_Caption(EXSCROLLBARLib.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.ScrollBar1
   .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(oScrollBar)
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.EXSCROLLBARLib.PartEnum Part)
{
}
```

VB

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

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

C#

```
private void ClickingPart(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ClickingPartEvent e)
{
}
```

C++

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

C++ Builder

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

Delphi

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

```
end;
```

Delphi 8 (.NET only)

```
procedure ClickingPart(sender: System.Object; e:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_ClickingPartEvent) Handles ClickingPart
End Sub
```

VB6

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

VBA

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

VFP

```
LPARAMETERS Part
```

Xbas…

```
PROCEDURE OnClickingPart(oScrollBar,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.ScrollBar.1::PartEnum)
end function
```

dBASE

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

The following VB sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub ScrollBar1_ClickPart(ByVal Part As EXSCROLLBARLibCtl.PartEnum)
   If (Part = exLeftB1Part) Then
      MsgBox ("Click")
   End If
End Sub
```

The following VB.NET sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub AxScrollBar1_ClickPart(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent) Handles AxScrollBar1.ClickPart
   If (e.part = EXSCROLLBARLib.PartEnum.exLeftB1Part) Then
      MsgBox("Click")
   End If
End Sub
```

The following C++ sample displays a message when the user clicks the exLeftB1Part part of the control:

```
void OnClickPartScrollbar1(long Part)
{
```

```
    if ( Part == 32768 /*exLeftB1Part*/ )
        MessageBox( "Click" );
}
```

The following C# sample displays a message when the user clicks the exLeftB1Part part of the control:

```
private void axScrollBar1_ClickPart(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent e)
{
    if (e.part == EXSCROLLBARLib.PartEnum.exLeftB1Part)
        MessageBox.Show("Click");
}
```

The following VFP sample displays a message when the user clicks the exLeftB1Part part of the control:

```
*** ActiveX Control Event ***
LPARAMETERS part

if ( part = 32768 )
    wait window "click"
endif
```

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. For instance, the VisiblePart(exLeftB1Part or exLeftB2Part) = True adds two new buttons left/up to the control.

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

C#

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

VB

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

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

C#

```
private void ClickPart(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent e)
{
}
```

C++

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

C++ Builder

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

Delphi

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

```
end;
```

Delphi 8 (.NET only)

```
procedure ClickPart(sender: System.Object; e:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent) Handles ClickPart
End Sub
```

VB6

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

VBA

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

VFP

```
LPARAMETERS Part
```

Xbas…

```
PROCEDURE OnClickPart(oScrollBar,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.ScrollBar.1::PartEnum)
end function
```

dBASE

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

The following VB sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub ScrollBar1_ClickPart(ByVal Part As EXSCROLLBARLibCtl.PartEnum)
   If (Part = exLeftB1Part) Then
      MsgBox ("Click")
   End If
End Sub
```

The following VB.NET sample displays a message when the user clicks the exLeftB1Part part of the control:

```
Private Sub AxScrollBar1_ClickPart(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent) Handles AxScrollBar1.ClickPart
   If (e.part = EXSCROLLBARLib.PartEnum.exLeftB1Part) Then
      MsgBox("Click")
   End If
End Sub
```

The following C++ sample displays a message when the user clicks the exLeftB1Part part of the control:

```
void OnClickPartScrollbar1(long Part)
{
```

```
    if ( Part == 32768 /*exLeftB1Part*/ )
        MessageBox( "Click" );
}
```

The following C# sample displays a message when the user clicks the exLeftB1Part part of the control:

```
private void axScrollBar1_ClickPart(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_ClickPartEvent e)
{
    if (e.part == EXSCROLLBARLib.PartEnum.exLeftB1Part)
        MessageBox.Show("Click");
}
```

The following VFP sample displays a message when the user clicks the exLeftB1Part part of the control:

```
*** ActiveX Control Event ***
LPARAMETERS part

if ( part = 32768 )
    wait window "click"
endif
```

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,
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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.ScrollBar.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.ScrollBar.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 ScrollBar1_DblClick(Shift As Integer, X As Single, Y As Single)
    Debug.Print ScrollBar1.PartFromPoint(X / Screen.TwipsPerPixelX, Y /
Screen.TwipsPerPixelY)
End Sub
```

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

```
Private Sub AxScrollBar1_DblClick(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_DblClickEvent) Handles AxScrollBar1.DblClick
    Debug.WriteLine(AxScrollBar1.get_PartFromPoint(e.x, e.y).ToString())
End Sub
```

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

```
void OnDblClickScrollbar1(short Shift, long X, long Y)
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_scrollbar.GetPartFromPoint(X, Y ) );
   OutputDebugString( strFormat );
}
```

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

```
private void axScrollBar1_DblClick(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_DblClickEvent e)
{
   System.Diagnostics.Debug.WriteLine(axScrollBar1.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.scrollbar1.PartFromPoint(x,y)))
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Use the EventParam(-2) to display entire information about fired event (such as name, identifier, and properties).

The Event notification occurs ANY time the control fires an event.

This is useful for X++ language, which does not support event with parameters passed by reference.

In X++ the "Error executing code: FormActiveXControl (data source), method ... called with invalid parameters" occurs when handling events that have parameters passed by reference. Passed by reference, means that in the event handler, you can change the value for that parameter, and so the control will takes the new value, and use it. The X++ is NOT able to handle properly events with parameters by reference, so we have the solution.

The solution is using and handling the Event notification and EventParam method., instead handling the event that gives the "invalid parameters" error executing code.

Let's presume that we need to handle the BarParentChange event to change the _Cancel parameter from false to true, which fires the "Error executing code: FormActiveXControl (data source), method onEvent_BarParentChange called with invalid parameters." We need to know the identifier of the BarParentChange event (each event has an unique identifier and it is static, defined in the control's type library). If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   print exscrollbar1.EventParam(-2).toString();
}
```

This code allows you to display the information for each event of the control being fired as in the list bellow:

```
"MouseMove/-606( 1 , 0 , 145 , 36 )" VT_BSTR
"BarParentChange/125( 192998632 , 'B' , 192999592 , =false )" VT_BSTR
"BeforeDrawPart/54( 2 , -1962866148 , =0 , =0 , =0 , =0 , =false )" VT_BSTR
```

```
"AfterDrawPart/55( 2 , -1962866148 , 0 , 0 , 0 , 0 )" VT_BSTR
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

Each line indicates an event, and the following information is provided: the name of the event, its identifier, and the list of parameters being passed to the event. The parameters that starts with = character, indicates a parameter by reference, in other words one that can changed during the event handler.

Now, we can see that the identifier for the BarParentChange event is 125, so we need to handle the Event event as:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   ;
   if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem as HITEM, Cancel as Boolean) */
      exscrollbar1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

The code checks if the BarParentChange (_EventID == 125) event is fired, and changes the third parameter of the event to true. The definition for BarParentChange event can be consulted in the control's documentation or in the ActiveX explorer. So, anytime you need to access the original parameters for the event you should use the EventParam method that allows you to get or set a parameter. If the parameter is not passed by reference, you can not change the parameter's value.

Now, let's add some code to see a complex sample, so let's say that we need to prevent moving the bar from an item to any disabled item. So, we need to specify the Cancel parameter as not Items.EnableItem(NewItem), in other words cancels if the new parent is disabled. Shortly the code will be:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   ;
   if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem as HITEM, Cancel as Boolean) */
      if ( !exscrollbar1.Items().EnableItem( exscrollbar1.EventParam( 2 /*NewItem*/ ) ) )
         exscrollbar1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

In conclusion, anytime the X++ fires the "invalid parameters." while handling an event, you can use and handle the Event notification and EventParam methods of the control

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

C#
```
private void Event(object sender,int EventID)
{
}
```

VB
```
Private Sub Event(ByVal sender As System.Object,ByVal EventID As Integer)
Handles Event
End Sub
```

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

C#
```
private void Event(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_EventEvent e)
{
}
```

C++
```
void OnEvent(long EventID)
{
}
```

C++ Builder
```
void __fastcall Event(TObject *Sender,long EventID)
{
}
```

Delphi
```
procedure Event(ASender: TObject; EventID : Integer);
begin
end;
```

Delphi 8 (.NET only)
```
procedure Event(sender: System.Object; e:
AxEXSCROLLBARLib._IScrollBarEvents_EventEvent);
begin
end;
```

Powe…
```
begin event Event(long EventID)
end event Event
```

VB.NET

```
Private Sub Event(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_EventEvent) Handles Event
End Sub
```

VB6

```
Private Sub Event(ByVal EventID As Long)
End Sub
```

VBA

```
Private Sub Event(ByVal EventID As Long)
End Sub
```

VFP

```
LPARAMETERS EventID
```

Xbas…

```
PROCEDURE OnEvent(oScrollBar,EventID)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComEvent Integer llEventID
   Forward Send OnComEvent llEventID
End_Procedure
```

Visual Objects

```
METHOD OCX_Event(EventID) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Event(int _EventID)
{
}
```

XBasic

```
function Event as v (EventID as N)
end function
```

dBASE

```
function nativeObject_Event(EventID)
return
```

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,
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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,
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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,
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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,
```

```
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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.ScrollBar.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.ScrollBar.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 ScrollBar1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   With ScrollBar1
      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 ScrollBar1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   ScrollBar1.Value = ScrollBar1.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 AxScrollBar1_MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent) Handles AxScrollBar1.MouseDownEvent
   With AxScrollBar1
      If (0 <> (.get_PartFromPoint(-1, -1) And EXSCROLLBARLib.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 AxScrollBar1_MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent) Handles AxScrollBar1.MouseDownEvent
   AxScrollBar1.Value = AxScrollBar1.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 OnMouseDownScrollbar1(short Button, short Shift, long X, long Y)
{
   if ( m_scrollbar.GetPartFromPoint(-1,-1) & 640 )
      m_scrollbar.SetValue( m_scrollbar.GetValueFromPoint(-1,-1) );
}
```

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

```
void OnMouseDownScrollbar1(short Button, short Shift, long X, long Y)
{
   m_scrollbar.SetValue( m_scrollbar.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 axScrollBar1_MouseDownEvent(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent e)
{
   if ( 0 != ( axScrollBar1.get_PartFromPoint(-1,-1) &
EXSCROLLBARLib.PartEnum.exBackgroundPart ) )
      axScrollBar1.Value = axScrollBar1.get_ValueFromPoint(-1, -1);
}
```

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

```
private void axScrollBar1_MouseDownEvent(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_MouseDownEvent e)
{
   axScrollBar1.Value = axScrollBar1.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.scrollbar1
   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.scrollbar1
   .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,
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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.ScrollBar.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.ScrollBar.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 ScrollBar1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
```

```
Single)
   Debug.Print ScrollBar1.PartFromPoint(X / Screen.TwipsPerPixelX, Y /
Screen.TwipsPerPixelY)
End Sub
```

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

```
Private Sub AxScrollBar1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXSCROLLBARLib._IScrollBarEvents_MouseMoveEvent) Handles
AxScrollBar1.MouseMoveEvent
   Debug.WriteLine(AxScrollBar1.get_PartFromPoint(e.x, e.y).ToString())
End Sub
```

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

```
void OnMouseMoveScrollbar1(short Button, short Shift, long X, long Y)
{
   CString strFormat;
   strFormat.Format( _T("%i"), m_scrollbar.GetPartFromPoint(X, Y ) );
   OutputDebugString( strFormat );
}
```

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

```
private void axScrollBar1_MouseMoveEvent(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_MouseMoveEvent e)
{
   System.Diagnostics.Debug.WriteLine(axScrollBar1.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.scrollbar1.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,
```

```
AxEXSCROLLBARLib._IScrollBarEvents_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_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(oScrollBar,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.ScrollBar.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.ScrollBar.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. For instance, the VisiblePart(exLeftB1Part or exLeftB2Part) = True adds two new buttons left/up to the control.

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

C#

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

VB

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

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

C#

```
private void OwnerDrawEnd(object sender,
AxEXSCROLLBARLib._IScrollBarEvents_OwnerDrawEndEvent e)
{
}
```

C++

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

C++ Builder

```
void __fastcall OwnerDrawEnd(TObject *Sender,Exscrollbarlib_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
AxEXSCROLLBARLib._IScrollBarEvents_OwnerDrawEndEvent) Handles
OwnerDrawEnd
End Sub
```

VB6

```
Private Sub OwnerDrawEnd(ByVal Part As EXSCROLLBARLibCtl.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(oScrollBar,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.ScrollBar.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 ScrollBar1
   .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 ScrollBar1_OwnerDrawEnd(ByVal Part As EXSCROLLBARLibCtl.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_scrollbar.SetOwnerDrawPart( 128 /*exUpperBackPart*/,  TRUE );
m_scrollbar.SetOwnerDrawPart( 512 /*exLowerBackPart*/,  TRUE );
```

```
void OnOwnerDrawEndScrollbar1(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. For instance, the VisiblePart(exLeftB1Part or exLeftB2Part) = True adds two new buttons left/up to the control.

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

C#

```
private void OwnerDrawStart(object sender,exontrol.EXSCROLLBARLib.PartEnum
Part,int hDC,ref bool DefaultPainting)
```

```
{
}
```

VB

```
Private Sub OwnerDrawStart(ByVal sender As System.Object,ByVal Part As
exontrol.EXSCROLLBARLib.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,
AxEXSCROLLBARLib._IScrollBarEvents_OwnerDrawStartEvent e)
{
}
```

C++

```
void OnOwnerDrawStart(long Part,long hDC,BOOL FAR* DefaultPainting)
{
}
```

C++ Builder

```
void __fastcall OwnerDrawStart(TObject *Sender,Exscrollbarlib_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:
AxEXSCROLLBARLib._IScrollBarEvents_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
```

```
AxEXSCROLLBARLib._IScrollBarEvents_OwnerDrawStartEvent) Handles
OwnerDrawStart
End Sub
```

VB6

```
Private Sub OwnerDrawStart(ByVal Part As EXSCROLLBARLibCtl.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(oScrollBar,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.ScrollBar.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 ScrollBar1
   .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 ScrollBar1_OwnerDrawEnd(ByVal Part As EXSCROLLBARLibCtl.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_scrollbar.SetOwnerDrawPart( 128 /*exUpperBackPart*/,  TRUE );
m_scrollbar.SetOwnerDrawPart( 512 /*exLowerBackPart*/,  TRUE );
```

```
void OnOwnerDrawEndScrollbar1(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;
      }
    }
}
```

Expressions

An expression is a string which defines a formula or criteria, that's evaluated at runtime. The expression may be a combination of variables, constants, strings, dates and operators/functions. For instance 1000 format `` gets 1,000.00 for US format, while 1.000,00 is displayed for German format.

The Exontrol's **eXPression** component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

Usage examples:

- 100 + 200, adds numbers and returns 300
- "100" + 200, concatenates the strings, and returns "100200"
- currency(1000) displays the value in currency format based on the current regional setting, such as "$1,000.00" for US format.
- 1000 format `` gets 1,000.00 for English format, while 1.000,00 is displayed for German format
- 1000 format `2|.|3|,` always gets 1,000.00 no matter of settings in the control panel.
- upper("string") converts the giving string in uppercase letters, such as "STRING"
- date(dateS('3/1/' + year(9:=#1/1/2018#)) + ((1:=(((255 - 11 * (year(=:9) mod 19)) - 21) mod 30) + 21) + (=:1 > 48 ? -1 : 0) + 6 - ((year(=:9) + int(year(=:9) / 4)) + =:1 + (=:1 > 48 ? -1 : 0) + 1) mod 7)) returns the date the Easter Sunday will fall, for year 2018. In this case the expression returns #4/1/2018#. If #1/1/2018# is replaced with #1/1/2019#, the expression returns #4/21/2019#.

Listed bellow are all predefined constants, operators and functions the general-expression supports:

The constants can be represented as:

- numbers in **decimal** format (where dot character specifies the decimal separator). For instance: -1, 100, 20.45, .99 and so on
- numbers in **hexa-decimal** format (preceded by **0x** or **0X** sequence), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, 0xFF,

0x00FF00, and so so.

- **date-time** in format **#mm/dd/yyyy hh:mm:ss#**, For instance, #1/31/2001 10:00# means the January 31th, 2001, 10:00 AM
- **string**, if it starts / ends with any of the **'** or ` or **"** characters. If you require the starting character inside the string, it should be escaped ( preceded by a \ character ). For instance, `Mihai`, "Filimon", 'has', "\"a quote\"", and so on

The predefined constants are:

- **bias** (BIAS constant), defines the difference, in minutes, between Coordinated Universal Time (UTC) and local time. For example, Middle European Time (MET, GMT+01:00) has a time zone bias of "-60" because it is one hour ahead of UTC. Pacific Standard Time (PST, GMT-08:00) has a time zone bias of "+480" because it is eight hours behind UTC. For instance, date(value - bias/24/60) converts the UTC time to local time, or date(date('now') + bias/24/60) converts the current local time to UTC time. For instance, "date(value - bias/24/60)" converts the value date-time from UTC to local time, while "date(value + bias/24/60)" converts the local-time to UTC time.
- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpi returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpix returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

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 binary range operators, all these with the same priority 5, are :

- a **MIN** b (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression value MIN 10 returns always a value greater than 10.
- a **MAX** b (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression value MAX 100 returns always a value less than 100.

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

- **:= (Store operator),** stores the result of expression to variable. The syntax for := operator is

  variable := expression

  where variable is a integer between 0 and 9. You can use the **=:** operator to restore any stored variable (please make the difference between := and =:). For instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

- **=: (Restore operator),** restores the giving variable (previously saved using the store operator). The syntax for =: operator is

  =: variable

  where variable is a integer between 0 and 9. You can use the **:=** operator to store the value of any expression (please make the difference between := and =:). For

instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

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 *?* operator 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. The type operator may return any of the following: 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. For instance *type(%1) = 8* specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, hex(`FF`) returns 255, while the hex(255) or hex(0xFF) returns the `FF` string. The hex(hex(`FFFFFFFF`)) always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitand 0x10111000 returns 0x00001000.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitor 0x10111000 returns 0x11111000.
- a **bitxor** b (binary operator) computes the XOR (exclusive-OR) operation on bits of a and b, and returns the unsigned value. For instance, 0x01110010 bitxor 0x10101010 returns 0x11011000.
- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, 128 bitshift 1 returns 64 (dividing by 2) or 128 bitshift (-1) returns 256 (multiplying by

2)

- **bitnot** (unary operator) flips every bit of x, and returns the unsigned value. For instance, bitnot(0x00FF0000) returns 0xFF00FFFF.

The operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as $1,000.00, for US format.
- 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

The operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"
- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahiM"
- a **startwith** b (binary operator) specifies whether a string starts with specified string (

0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1

- a **endwith** b (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- a **contains** b (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- a **left** b (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- a **right** b (binary operator) retrieves the right part of the string. For instance *"Mihai" right 2* returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" lfind "C"* returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" rfind "C"* returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance *"Mihai" mid 2* returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance *"Mihai" count "i"* returns 2.
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result. For instance, the *"Mihai" replace "i" with ""* returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b (binary operator) 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.
- a **like** b (binary operator) compares the string a against the pattern b. The pattern b may contain wild-characters such as *, ?, # or [] and can have multiple patterns separated by space character. In order to have the space, or any other wild-character inside the pattern, it has to be escaped, or in other words it should be preceded by a \ character. For instance value like `F*e` matches all strings that start with F and ends on e, or value like `a* b*` indicates any strings that start with a or b character.
- a **lpad** b (binary operator) pads the value of a to the left with b padding pattern. For instance, 12 lpad "0000" generates the string "0012".
- a **rpad** b (binary operator) pads the value of a to the right with b padding pattern. For instance, 12 lpad "____" generates the string "12__".
- a **concat** b (binary operator) concatenates the a (as string) for b times. For instance, "x" concat 5, generates the string "xxxxx".

The operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"

- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the *timeF(#1/1/2001 13:00#)* returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the *shortdate(#1/1/2001 13:00#)* returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the *shortdateF(#1/1/2001 13:00#)* returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the *dateF(#01/01/2001 14:00:00#)* returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the *longdate(#1/1/2001 13:00#)* returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the *year(#12/31/1971 13:14:15#)* returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the *month(#12/31/1971 13:14:15#)* returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the *day(#12/31/1971 13:14:15#)* returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the *yearday(#12/31/1971 13:14:15#)* returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the *weekday(#12/31/1971 13:14:15#)* returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the *hour(#12/31/1971 13:14:15#)* returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the *min(#12/31/1971 13:14:15#)* returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the *sec(#12/31/1971 13:14:15#)* returns 15

The expression supports also **immediate if** (similar with iif in visual basic, or ? : in C++) ie cond ? value_true : value_false, which means that once that cond is true the value_true is used, else the value_false is used. Also, it supports variables, up to 10 from 0 to 9. For instance, 0:="Abc" means that in the variable 0 is "Abc", and =:0 means retrieves the value of the variable 0. For instance, the len(%0) ? (0:=(%1+%2) ? currency(=:0) else `` ) : `` gets the sum between second and third column in currency format if it is not zero, and only if the first column is not empty. As you can see you can use the variables to avoid computing several times the same thing (in this case the sum %1 and %2 .