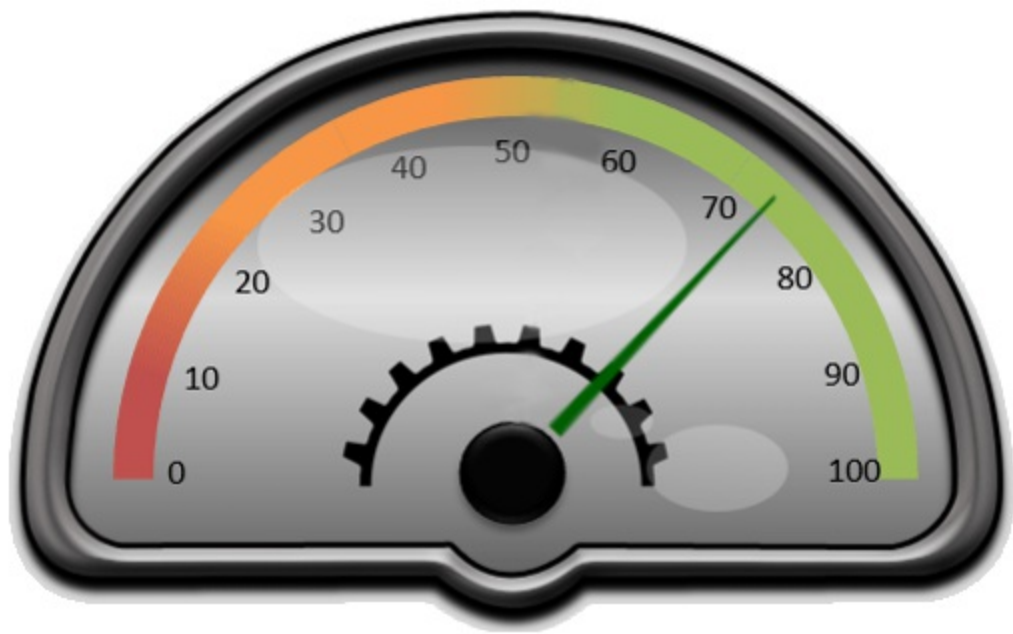


## ExGauge

The eXGauge / eXLayers library provides graphics capabilities to visually display and edit the amount, level, or contents of something. The view can show one or more layers, where each layer can display one or more transparent pictures, HTML captions which can be clipped, moved, rotated or combination of them, by dragging the mouse, rolling the mouse wheel, or using the keyboard. Using the eXGauge / eXLayers library you can easily simulate any gauges, thermometers, meters, clocks, buttons, sliders, scales, knobs, dials, switches, progress, status, indicators, LEDs, and so on. As usual, there are no dependencies to MFC, VB, VCL, or anything else.

Features include:

- Multiple Layers Support
- Ability to display the control's itself ( no form, transparent form, no background ) as an individual or child widget
- Any layer can display multiple graphics, images, HTML captions
- Ability to specify visible, selectable objects on any layer
- Any layer can be clipped, moved, rotated, or combination of any of these
- Clipping support include intersection of any of rectangle, round rectangle, ellipse, pie, picture mask, polygon, and so on
- Visibility / Transparency support
- Brightness, Contrast, Grayscale support
- Drag, Mouse-Wheel, Keyboard Support
- Mouse In, Mouse Out, Smooth Change Support
- High-Quality Rotation Support
- ToolTip support
- Debug Mode support, allows you to display debugging information
- Expression Support, for any angle, offset or clip's value



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

## How to get support?

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

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

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

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

Regards,  
Exontrol Development Team

<https://www.exontrol.com>

# constants AnchorEnum

The AnchorEnum type specifies how the object is anchored. The [Caption\(exLayerCaptionAnchor\)](#) / [ExtraCaption\(...,exLayerCaptionAnchor\)](#) / [Foreground.Caption\(exLayerCaptionAnchor\)](#) / [Foreground.ExtraCaption\(,exLayerCaptionAnchor\)](#) property specifies how the caption is anchored. The AnchorEnum type supports the following values:

Name	Value	Description
exAnchorDock	0	The object is anchored to the host's client area.
exAnchorTop	1	The object is anchored to the top side of its host.
exAnchorBottom	2	The object is anchored to the bottom side of its host.
exAnchorLeft	4	The object is anchored to the left side of its host.
exAnchorRight	8	The object is anchored to the right side of its host.

# constants AppearanceEnum

The AppearanceEnum type specifies how the control's border are shown. The [Appearance](#) property specifies the control's border. The Appearance property supports the following values:

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

# constants BackgroundPartEnum

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

Name	Value	Description
exToolTipAppearance	64	Specifies the visual appearance of the borders of the tooltips. Use the <a href="#">ToolTipPopDelay</a> property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the <a href="#">ToolTip</a> / <a href="#">ToolTipTitle</a> property to specify the layer's tooltip. Use the <a href="#">ToolTipWidth</a> property to specify the width of the tooltip window. The <a href="#">ToolTipDelay</a> property specifies the time in ms that passes before the ToolTip appears. Use the <a href="#">ShowToolTip</a> method to display a custom tooltip.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.

# constants ColorAdjustmentChannelEnum

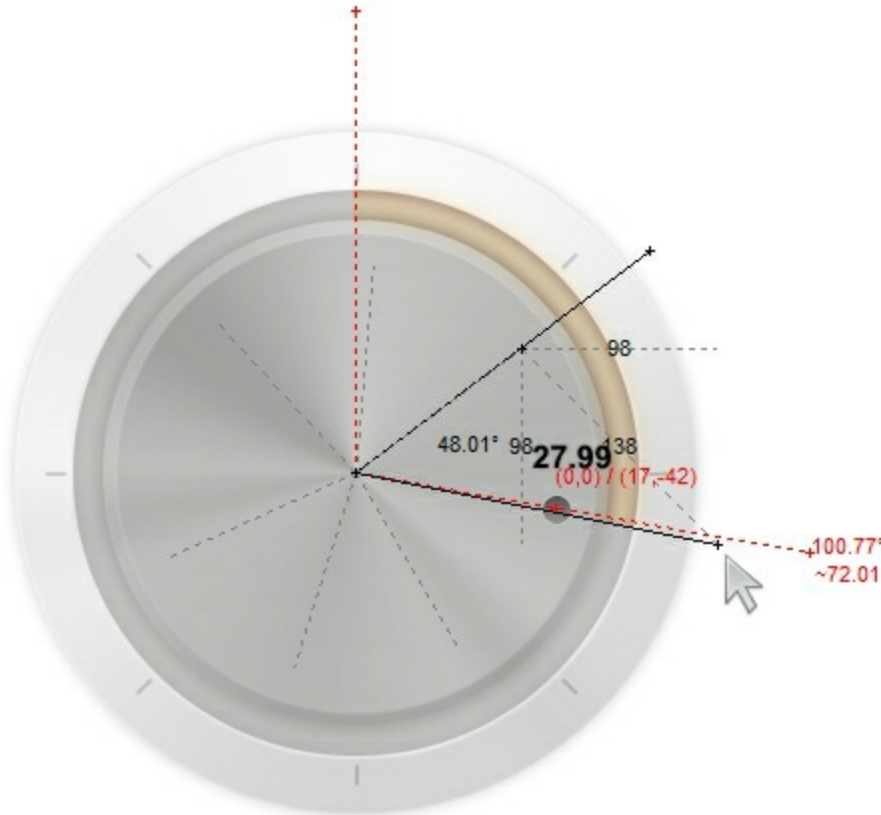
The ColorAdjustmentChannelEnum type specifies the color channel to be updated by the [Brightness](#) / [Contrast](#) properties. These properties can be used to change the percent of specified color to be applied on the layer. The ColorAdjustmentChannelEnum defines the following values:

Name	Value	Description
exAllChannels	0	The value is applied to all channels ( red, green and blue ).
exRedChannel	1	The value is applied to the red channel only
exGreenChannel	2	The value is applied to the green channel only
exBlueChannel	3	The value is applied to the blue channel only

## constants DebugLayerDragEnum

A `DebugLayerDragEnum` type that holds what information the debugging the drag operation should display. The [Debug](#) property specifies debugging information to be shown while dragging the layers.

The following information shows all debug information while dragging the layer:



The `DebugLayerDragEnum` type supports the following flags:

Name	Value	Description
exDebugLayerDragNothing	0	No debug information is displayed.
exDebugLayerDragHighlight	256	Specifies that layers should be shown with a semi-transparent color, so the debugging information is more clear.
exDebugLayerDragClick	1	Shows the point where the drag operation begins. The <a href="#">X</a> property indicates the x-position of the cursor, when the drag operation starts and the <a href="#">Y</a> property indicates the y-position of the cursor, when the drag operation starts.
		Shows the current dragging point. The <a href="#">CurrentX</a> property indicates the current x-position of the

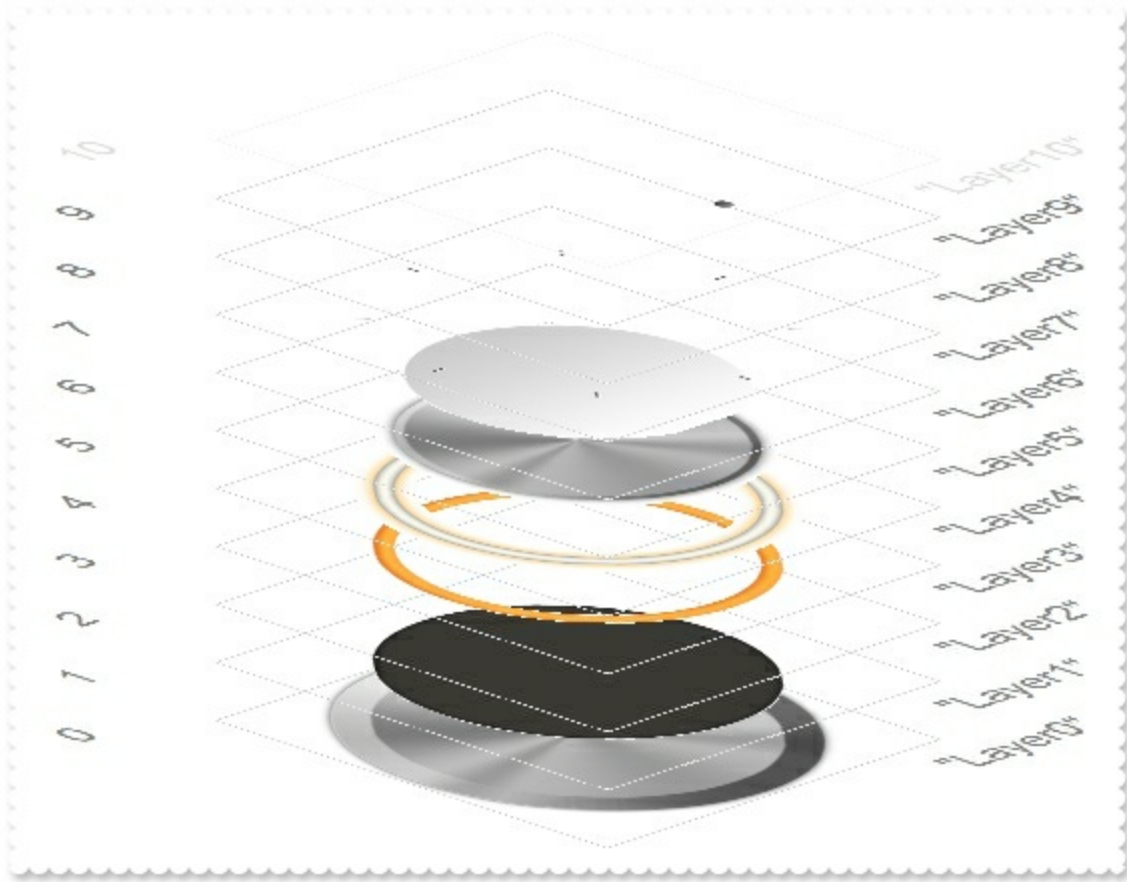


exDebugLayerDragCurrent	2	cursor, while dragging the layer and the <a href="#">CurrentY</a> property indicates the current y-position of the cursor, while dragging the layer.
exDebugLayerDragDeltaX	4	Shows the horizontal offset. The <a href="#">DeltaX</a> property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
exDebugLayerDragDeltaY	8	Shows the vertical offset. The <a href="#">DeltaY</a> property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.
exDebugLayerDragDelta	16	Shows the distance between clicking and current point. The <a href="#">Delta</a> property, returns the distance between clicking and current points.
exDebugLayerDragMove	287	Shortcut flag for move operation by dragging. It combines exDebugLayerDragClick + exDebugLayerDragCurrent + exDebugLayerDragDeltaX + exDebugLayerDragDeltaY + exDebugLayerDragDelta + exDebugLayerDragHighlight
exDebugLayerDragRotateCenter	32	Shows the rotation center of the dragging layer. The <a href="#">RotateCenterLayer</a> , <a href="#">RotateCenterX</a> and <a href="#">RotateCenterY</a> properties determines the (x,y) rotation center, relative to specified layer.
exDebugLayerDragDeltaAngle	64	Shows the delta angle. The <a href="#">DeltaAngle</a> property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.
exDebugLayerDragDeltaAngleMultiply	128	Adds lines to split equally the circle.
exDebugLayerDragRotate	483	Shortcut flag for rotate operation by dragging. It combines exDebugLayerDragClick + exDebugLayerDragCurrent + exDebugLayerDragRotateCenter + exDebugLayerDragDeltaAngle + exDebugLayerDragDeltaAngleMultiply + exDebugLayerDragHighlight.
exDebugLayerDragAll	-1	Includes all debug information.

## constants DebugLayerEnum

The DebugLayerEnum type indicates the values of Debug property. Use the [Debug](#) property to display the layers in debug mode.

The following screen shot shows the control while Debug property is exDebugLayers:



The DebugLayerEnum type supports the following properties and method:

Name	Value	Description
exNoDebugLayer	0	No debug information is shown.
exDebugLayers	1	The control shows all layers in debug mode. The <a href="#">ShowLayers</a> property indicates the only layers to be shown on the control. The exDebugLayers flag can be combined with the exDebugAutoScroll flag.
exDebugVisibleLayers	2	The control shows only visible layers in debug mode. The <a href="#">ShowLayers</a> property indicates the only layers to be shown on the control. The exDebugVisibleLayers flag can be combined with the exDebugAutoScroll flag
exDebugAutoScroll	256	The user can scroll the layers into the debug view.

# constants DefaultLayerPropertyEnum

The DefaultLayerPropertyEnum type specifies the properties of the layer, whose default value can be changed by the [DefaultLayer](#) property. Any call of the [DefaultLayer](#) property has effect for any new layer added to the control's collection. Changing the [DefaultLayer](#) property does not have any effect on existing layers. It does have effect on any new layer added to the control. The DefaultLayerPropertyEnum type supports the following values:

Name	Value	Description
exDefLayerVisible	0	Retrieves or sets a value indicating whether the layer is visible or hidden. Specifies the default value of the <a href="#">Visible</a> property.
exDefLayerSelectable	1	Returns or sets a value that indicates whether the layer is selectable. Specifies the default value of the <a href="#">Selectable</a> property.
exDefLayerLeft	16	Specifies the expression relative to the view, to determine the x-position to show the current layer on the control. Specifies the default value of the <a href="#">Left</a> property.
exDefLayerTop	17	Specifies the expression relative to the view, to determine the y-position to show the current layer on the control. Specifies the default value of the <a href="#">Top</a> property.
exDefLayerWidth	18	Specifies the expression relative to the view, to determine the width to show the current layer on the control. Specifies the default value of the <a href="#">Width</a> property.
exDefLayerHeight	19	Specifies the expression relative to the view, to determine the height to show the current layer on the control. Specifies the default value of the <a href="#">Height</a> property.
exDefLayerToolTip	20	Gets or sets a value (HTML tooltip) that's displayed once the cursor hovers the layer. Specifies the default value of the <a href="#">ToolTip</a> property.
exDefLayerToolTipTitle	21	Gets or sets a value (title) that's displayed once the cursor hovers the layer. Specifies the default value of the <a href="#">ToolTipTitle</a> property.
exDefLayerTransparency	22	Gets or sets a value that indicates percent of the transparency to display the layer. Specifies the default value of the <a href="#">Transparency</a> property.

exDefLayerGrayscale	23	Returns or sets a value that indicates whether the layer is show as grayscale. Specifies the default value of the <a href="#">Grayscale</a> property.
exDefLayerUserData	24	Indicates any extra data associated with the layer. Specifies the default value of the <a href="#">UserData</a> property.
exDefLayerBrightness	128	Specifies the percent of brightness to apply to the layer ( on all channels ). Specifies the default value of the <a href="#">Brightness(exAllChannels)</a> property.
exDefLayerBrightnessRed	129	Specifies the percent of brightness to apply to the layer ( on red channel ). Specifies the default value of the <a href="#">Brightness(exRedChannel)</a> property.
exDefLayerBrightnessGreen	130	Specifies the percent of brightness to apply to the layer ( on green channel ). Specifies the default value of the <a href="#">Brightness(exGreenChannel)</a> property.
exDefLayerBrightnessBlue	131	Specifies the percent of brightness to apply to the layer ( on blue channel ). Specifies the default value of the <a href="#">Brightness(exBlueChannel)</a> property.
exDefLayerContrast	144	Specifies the percent of contrast to apply to the layer ( on all channels ). Specifies the default value of the <a href="#">Contrast(exAllChannels)</a> property.
exDefLayerContrastRed	145	Specifies the percent of contrast to apply to the layer ( on red channel ). Specifies the default value of the <a href="#">Contrast(exRedChannel)</a> property.
exDefLayerContrastGreen	146	Specifies the percent of contrast to apply to the layer ( on green channel ). Specifies the default value of the <a href="#">Contrast(exGreenChannel)</a> property.
exDefLayerContrastBlue	147	Specifies the percent of contrast to apply to the layer ( on blue channel ). Specifies the default value of the <a href="#">Contrast(exBlueChannel)</a> property.
exDefLayerOffsetX	160	Gets or sets a value that indicates x-offset of the layer. Specifies the default value of the <a href="#">OffsetX</a> property.
exDefLayerOffsetY	161	Gets or sets a value that indicates x-offset of the layer. Specifies the default value of the <a href="#">OffsetX</a> property.
exDefLayerDefaultOffsetX	162	Gets or sets a value that indicates the default x-offset of the layer. Specifies the default value of the

[DefaultOffsetX](#) property.

exDefLayerDefaultOffsetY	163	Gets or sets a value that indicates the default y-offset of the layer. Specifies the default value of the <a href="#">DefaultOffsetY</a> property.
--------------------------	-----	--

exDefLayerOffsetXValid	164	Validates the x-offset value of the layer. Specifies the default value of the <a href="#">OffsetXValid</a> property.
------------------------	-----	--

exDefLayerOffsetYValid	165	Validates the y-offset value of the layer. Specifies the default value of the <a href="#">OffsetYValid</a> property.
------------------------	-----	--

exDefLayerValueToOffsetX	166	Specifies the expression to convert the value to x-offset. Specifies the default value of the <a href="#">ValueToOffsetX</a> property.
--------------------------	-----	--

exDefLayerValueToOffsetY	167	Specifies the expression to convert the value to y-offset. Specifies the default value of the <a href="#">ValueToOffsetY</a> property.
--------------------------	-----	--

exDefLayerOffsetToValue	168	Specifies the expression to convert the offsetx,offsety to value. Specifies the default value of the <a href="#">OffsetToValue</a> property.
-------------------------	-----	--

exDefLayerRotateAngle	176	Specifies the angle to rotate the layer. Specifies the default value of the <a href="#">RotateAngle</a> property.
-----------------------	-----	---

exDefLayerDefaultRotateAngle	177	Specifies the default angle to rotate the layer. Specifies the default value of the <a href="#">DefaultRotateAngle</a> property.
------------------------------	-----	--

exDefLayerRotateAngleValid	178	Validates the rotation angle of the layer. Specifies the default value of the <a href="#">RotateAngleValid</a> property.
----------------------------	-----	--

exDefLayerRotateCenterLayer	179	Indicates the index of the layer the rotation is around. If -1, the rotation is relative to the current layer. Specifies the default value of the <a href="#">RotateCenterLayer</a> property.
-----------------------------	-----	---

exDefLayerRotateCenterX	180	Indicates the expression that determines the x-origin of the rotation point relative to the <a href="#">RotateCenterLayer</a> layer. Specifies the default value of the <a href="#">RotateCenterX</a> property.
-------------------------	-----	---

exDefLayerRotateCenterY	181	Indicates the expression that determines the y-origin of the rotation point relative to the <a href="#">RotateCenterLayer</a> layer. Specifies the default value of the <a href="#">RotateCenterY</a> property.
-------------------------	-----	---

exDefLayerValueToRotateAngle	182	Specifies the expression to convert the value to rotating angle. Specifies the default value of the
------------------------------	-----	---

[ValueToRotateAngle](#) property.

exDefLayerRotateAngleToValue 183

Specifies the expression to convert the rotating angle to value. Specifies the default value of the [RotateAngleToValue](#) property.

exDefLayerRotateClip 184

Specifies whether the layer's clipping region is rotated once the layer is rotated. Specifies the default value of the [RotateClip](#) property.

exDefLayerRotateType 185

Returns or sets a value that indicates whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ... Specifies the default value of the [RotateType](#) property.

exDefLayerShowHandCursor 192

Returns or sets a value that indicates whether the hand cursor is shown when it hovers a visible / selectable / draggable layer. Specifies the default value of the [ShowHandCursor](#) property.

---

# constants LayerClipTypeEnum

The LayerClipTypeEnum type specifies the clipping types currently any layer can support. The [Type](#) property specifies the type of the clipping the current layer supports. The Type property can be any combination of the following flags, which indicates intersection of them. The LayerClipTypeEnum type supports the following value:

Name	Value	Description
exLayerClipEmpty	0	No clipping is applied to the layer.
exLayerClipRectangle	1	Indicates that the <a href="#">ClipRectangle</a> is applied on the layer.
exLayerClipRoundRectangle	2	Indicates that the <a href="#">ClipRoundRectangle</a> is applied on the layer.
exLayerClipEllipse	4	Indicates that the <a href="#">ClipEllipse</a> is applied on the layer.
exLayerClipPie	8	Indicates that the <a href="#">ClipPie</a> is applied on the layer.
exLayerClipPolygon	16	Indicates that the <a href="#">ClipPolygon</a> is applied on the layer.
exLayerClipPicture	32	Indicates that the <a href="#">ClipPicture</a> is applied on the layer.

# constants LayerUpdateEnum

The LayerUpdateEnum type specifies the way the control clips its content. The control support transparent form, or in other words, displaying the control's itself without its form behind.

Currently, the control supports two type of clippings:

- by region clipping, using the [LayerClipTo](#) property
- by layering, using the [LayerUpdate](#) property

The LayerUpdateEnum type supports the following values:

Name	Value	Description
exLayerUpdateControl	0	By default, the control updates its content.
exLayerUpdateParent	1	Updates the parent's device, to clip the control inside.
exLayerUpdateScreen	2	Updates the screen's device, to clip the control inside.



# constants OnDragLayerEnum

The OnDragLayerEnum type indicates the operation a layer can perform when user clicks and drags it. The [OnDrag](#) property indicates the action to be performed when the user drags the layer. The OnDragLayerEnum type supports the following value:

Name	Value	Description
exDoNothing	0	Nothing happens if the user drags the layer.
exDoMove	1	The layer is moved while dragging. The layer's <a href="#">OffsetX</a> and <a href="#">OffsetY</a> indicates the current (x,y) position of the layer.
exDoRotate	2	The layer is rotated while dragging. The <a href="#">RotateAngle</a> property indicates the currently rotation angle.
exDoRotamove	3	The layer is moved by rotation while dragging. The <a href="#">RotateAngle</a> property indicates the currently rotation angle. In this case, the layer's <a href="#">RotamoveOffsetX</a> / <a href="#">RotamoveOffsetY</a> property indicates the current (x,y) position of the layer.

# constants **PictureDisplayEnum**

Specifies how a picture object is displayed. The [DisplayAs](#) property retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background. The PictureDisplayEnum type supports the following values:

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

# constants PropertyLayerCaptionEnum

The PropertyLayerCaptionEnum type holds properties of the HTML caption that can be displayed on the control, or on the layer's foreground. Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The PropertyLayerCaptionEnum type supports the following value:

Name	Value	Description
exLayerCaption	0	Indicates the HTML caption to be displayed on the caption. By default, the exLayerCaption is empty. You can use the exLayerCaptionWordWrap to display the caption on multiple lines. The exLayerCaption supports built-in HTML format as listed <a href="#">here</a> .  (string expression)
exLayerCaptionBackColor	1	Indicates the layer's background color. By default, the exLayerCaptionBackColor property is -1, which indicates that no background color is applied. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used. You can use the <bgcolor> HTML tag in the exLayerCaption to specify a different background color for a portion of the text. Use the <a href="#">Add</a> method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.  (long expression)

Indicates the layer's foreground color. By default, the exLayerCaptionForeColor property is -1, which

exLayerCaptionForeColor	2	indicates that no foreground color is applied. You can use the <fgcolor> HTML tag in the exLayerCaption to specify a different foreground color for a portion of the text.
-------------------------	---	--

*(long expression)*

exLayerCaptionAnchor	3	Specifies the side of the host where the caption is anchored. By default, the exLayerCaptionAnchor property is 1 (exAnchorTop), that indicates that the caption is anchored to the top side of its host. You can use the exLayerCaptionLeft, exLayerCaptionTop, exLayerCaptionWidth and exLayerCaptionHeight to display the caption at a different position relative to its original position.
----------------------	---	--

*([AnchorEnum](#) type).*

Specifies the expression to determine the x-position to show the caption, relative to its current position. By default, the exLayerCaptionLeft property is "0", which indicates that the caption is displayed at it's original position (horizontal axis), determined by the exLayerCaptionAnchor. You can use the exLayerCaptionAnchor property to anchor the caption to a different side of the host.

The property supports the following keywords:

exLayerCaptionLeft	4	<ul style="list-style-type: none"> <li>• <b>twidth</b>, indicates the width required to fully display the caption</li> <li>• <b>theight</b>, indicates the height required to fully display the caption</li> <li>• <b>width</b>, indicates the width of the layer ( if it is applied to the layer's foreground <a href="#">Foreground.Caption</a> or <a href="#">Foreground.ExtraCaption</a> ), or of the control ( if it is applied to the control's foreground <a href="#">Caption</a> or <a href="#">ExtraCaption</a> )</li> <li>• <b>height</b>, indicates the height of the layer ( if it is applied to the layer's foreground <a href="#">Foreground.Caption</a> or <a href="#">Foreground.ExtraCaption</a> ), or of the control ( if</li> </ul>
--------------------	---	--

it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )

The property supports predefined constants, operators and functions as described [here](#) .

*(string expression)*

Specifies the expression to determine the y-position to show the caption, relative to its current position. By default, the exLayerCaptionTop property is "0", which indicates that the caption is displayed at it's original position (vertical axis), determined by the exLayerCaptionAnchor. You can use the exLayerCaptionAnchor property to anchor the caption to a different side of the host.

The property supports the following keywords:

- **twidht**, indicates the width required to fully display the caption
- **theight**, indicates the height required to fully display the caption
- **width**, indicates the width of the layer ( if it is applied to the layer's foreground [Foreground.Caption](#) or [Foreground.ExtraCaption](#) ), or of the control ( if it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )
- **height**, indicates the height of the layer ( if it is applied to the layer's foreground [Foreground.Caption](#) or [Foreground.ExtraCaption](#) ), or of the control ( if it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )

The property supports predefined constants, operators and functions as described [here](#) .

*(string expression)*

Specifies the expression to determine the width to show the caption, relative to its current width. By

exLayerCaptionTop

5

default, the `exLayerCaptionWidth` property is "twidth", which indicates that the caption is displayed on its full width. You can use the `exLayerCaptionAnchor` property to anchor the caption to a different side of the host.

The property supports the following keywords:

`exLayerCaptionWidth`

6

- **twidth**, indicates the width required to fully display the caption
- **theight**, indicates the height required to fully display the caption
- **width**, indicates the width of the layer ( if it is applied to the layer's foreground [Foreground.Caption](#) or [Foreground.ExtraCaption](#) ), or of the control ( if it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )
- **height**, indicates the height of the layer ( if it is applied to the layer's foreground [Foreground.Caption](#) or [Foreground.ExtraCaption](#) ), or of the control ( if it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )

The property supports predefined constants, operators and functions as described [here](#) .

*(string expression)*

Specifies the expression to determine the height to show the caption, relative to its current height. By default, the `exLayerCaptionHeight` property is "theight", which indicates that the caption is displayed on its full height. You can use the `exLayerCaptionAnchor` property to anchor the caption to a different side of the host.

The property supports the following keywords:

- **twidth**, indicates the width required to fully display the caption
- **theight**, indicates the height required to fully display the caption

exLayerCaptionHeight

7

- **width**, indicates the width of the layer ( if it is applied to the layer's foreground [Foreground.Caption](#) or [Foreground.ExtraCaption](#) ), or of the control ( if it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )
- **height**, indicates the height of the layer ( if it is applied to the layer's foreground [Foreground.Caption](#) or [Foreground.ExtraCaption](#) ), or of the control ( if it is applied to the control's foreground [Caption](#) or [ExtraCaption](#) )

The property supports predefined constants, operators and functions as described [here](#) .

*(string expression)*

exLayerCaptionWordWrap

8

Indicates whether a multiline caption automatically wraps words to the beginning of the next line when necessary. By default, the exLayerCaptionWordWrap property is False.

*(boolean expression)*

exLayerCaptionBackgroundEx0

Indicates Unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the layer's background, using EBN String Format. A short description of the EBN String Format is described [here](#), or a full description of the EBN String Format can be found [here](#).

*(string expression)*

exLayerCaptionVisibleFront

10

Specifies whether the caption is shown in front. By default, the exLayerCaptionVisibleFront property is True, which indicates that the caption is shown in front. Use the exLayerCaptionVisibleFront property to display the caption on the layer's background, if the exLayerCaptionVisibleFront property is False.

*(boolean expression)*

The exLayerCaption supports built-in HTML tags as follow:

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**<font ;12>bit</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part



of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.

- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **<b>bold</b>** in HTML caption you can use **&lt;b>&gt;bold&lt;/b>**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The **<font>** HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The **<gra>** with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "**<font ;18><gra FFFFFFFF;1;1>**gradient-center**</gra></font>**" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or **<fgcolor>** defines the color to show the inside text. The **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><out 000000>**  
**<fgcolor=FFFFFF>**outlined**</fgcolor></out></font>**" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or **<fgcolor>** defines the

color to show the inside text. The `<font>` HTML tag can be used to define the height of the font. For instance the "`<font ;31><sha>shadow</sha></font>`" generates the following picture:

shadow

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

outline anti-aliasing

The property supports predefined constants, operators and functions as listed bellow:

*The constants are ( DPI-Aware components ):*

- **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 y-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** ( remainder 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 :*

- **MIN** ( 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.
- **MAX** ( 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 than *iif* (immediate if operator) alternative.

- **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 than using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is

determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

- 0 - empty ( not initialized )
  - 1 - null
  - 2 - short
  - 3 - long
  - 4 - float
  - 5 - double
  - 6 - currency
  - 7 - date
  - 8 - string
  - 9 - object
  - 10 - error
  - 11 - boolean
  - 12 - variant
  - 13 - any
  - 14 - decimal
  - 16 - char
  - 17 - byte
  - 18 - unsigned short
  - 19 - unsigned long
  - 20 - long on 64 bits
  - 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"
- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startswith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1



- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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 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.

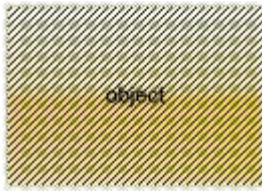
# The EBN String Format syntax in BNF notation is defined like follows:

```
<EBN> ::= <elements> | <root> "(" [<elements>] ")"
<elements> ::= <element> [ "," <elements> ]
<root> ::= "root" [ <attributes> ] | [ <attributes> ]
<element> ::= <anchor> [ <attributes> ] [ "(" [<elements>] ")" ]
<anchor> ::= "none" | "left" | "right" | "client" | "top" | "bottom"
<attributes> ::= "[" [<client> "," <attribute> [ "," <attributes> ] "]"
<client> ::= <expression> | <expression> "," <expression> "," <expression> ","
<expression>
<expression> ::= <number> | <number> "%"
<attribute> ::= <bgcolor> | <text> | <wordwrap> | <align> | <pattern> |
<patterncolor> | <frame> | <framethick> | <data> | <others>
<equal> ::= "="
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<decimal> ::= <digit> <decimal>
<hexadigit> ::= <digit> | "A" | "B" | "C" | "D" | "E" | "F"
<hexa> ::= <hexadigit> <hexa>
<number> ::= <decimal> | "0x" <hexa>
<color> ::= <rgbcolor> | number
<rgbcolor> ::= "RGB" "(" <number> "," <number> "," <number> ")"
<string> ::= "\"" <characters> "\"" | "\"" <characters> "\"" | " " <characters> "
<characters> ::= <char> | <characters>
<char> ::= <any_character_excepts_null>
<bgcolor> ::= "back" <equal> <color>
<text> ::= "text" <equal> <string>
<align> ::= "align" <equal> <number>
<pattern> ::= "pattern" <equal> <number>
<patterncolor> ::= "patterncolor" <equal> <color>
<frame> ::= "frame" <equal> <color>
<data> ::= "data" <equal> <number> | <string>
<framethick> ::= "framethick"
<wordwrap> ::= "wordwrap"
```

*Others like: pic, stretch, hstretch, vstretch, transparent, from, to are reserved for future use only.*

## Easy samples:

- "[pattern=6]", shows the BDiagonal pattern on the object's background.



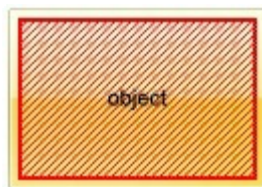
- "[frame=RGB(255,0,0),framethick]", draws a red thick-border around the object.



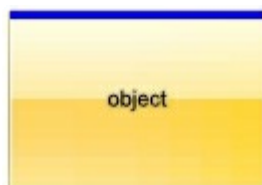
- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a red thick-border around the object, with a patten inside.



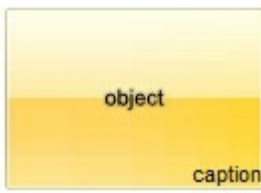
- "[[patterncolor=RGB(255,0,0)]  
(none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(255,0,0),framethick])]", draws a red thick-border around the object, with a patten inside, with a 4-pixels wide padding:



- "top[4,back=RGB(0,0,255)]", draws a blue line on the top side of the object's background, of 4-pixels wide.



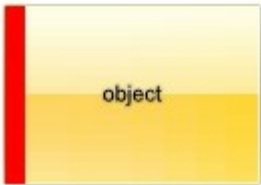
- "[text=`caption`,align=0x22]", shows the caption string aligned to the bottom-right side of the object's background.



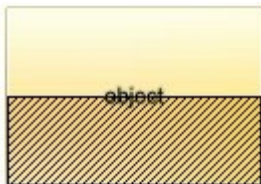
- "[text=`<img>flag</img>`,align=0x11]" shows the flag picture and the sweden string aligned to the bottom side of the object.



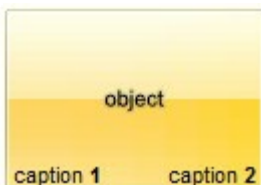
- "left[10,back=RGB(255,0,0)]", draws a red line on the left side of the object's background, of 10-pixels wide.



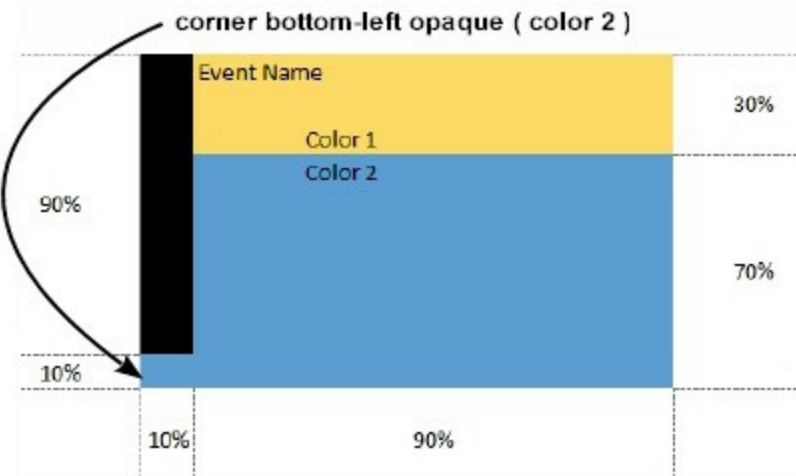
- "bottom[50%,pattern=6,frame]", shows the BDiagonal pattern with a border around on the lower-half part of the object's background.



- "root[text=`caption <b>2` ,align=0x22](client[text=`caption <b>1` ,align=0x20])", shows the caption **1** aligned to the bottom-left side, and the caption **2** to the bottom-right side



Now, lets say we have the following request to layout the colors on the objects:



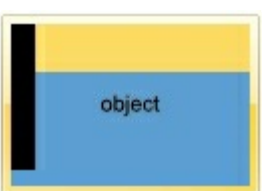
We define the BackgroundExt property such as "top[30%,back=RGB(253,218,101)],client[back=RGB(91,157,210)],none[(0%,0%,10%,100%)(top[90%,back=RGB(0,0,0)])]", and it looks as:

To String: `top[30%,back=RGB(253,218,101)],client[back=RGB(91,157,210)],none[(0%,0%,10%,100%)](top[90%,back=RGB(0,0,0)])`

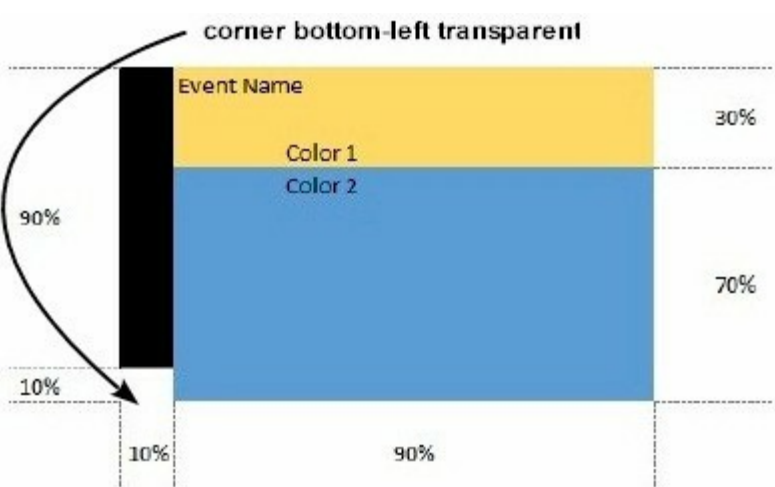
Diagram showing the layout of the UI component. The layout is defined by the following properties:

- Root
- Top<sub>30%</sub>
- Client
- None<sub>0%,0%,10%,100%</sub>
- Top<sub>90%</sub>

so, if we apply to our object we got:

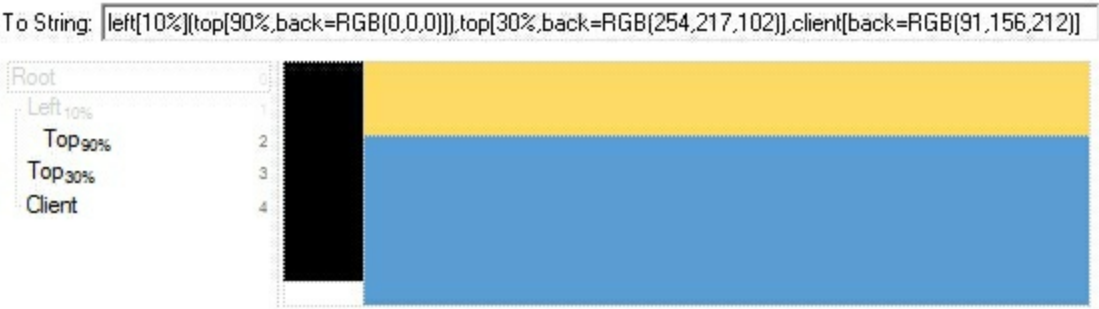


Now, lets say we have the following request to layout the colors on the objects:

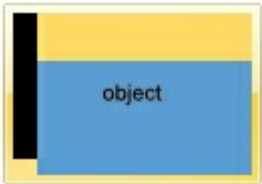


We define BackgroundExt property such as "left[10%]

(top[90%,back=RGB(0,0,0)]),top[30%,back=RGB(254,217,102)],client[back=RGB(91,156,212)] and it looks as:



so, if we apply to our object we got:



# constants RotateTypeEnum

The RotateTypeEnum type indicates the type of rotation currently, the control supports. The [RotateType](#) property returns or sets a value that indicates whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ... The RotateTypeEnum type supports the following values.

Name	Value	Description
exRotateFast	0	This is the default value. It is the fastest method compared with others but the images is not as smooth as possible.
exRotateByShear	1	The method also called "Rotation Through Shearing", unlike traditional rotation of images, where every n'th pixel is sampled and copied to the result image, this template provides much more accurate image rotation features (weighing the pixels).
exRotateBilinearInterpolation	2	This method also called "Rotation by Bilinear Interpolation", is fast, and produces perfect rotation images.



# constants SmoothPropertyEnum

The SmoothPropertyEnum type specifies the properties of the layer that can be changed gradually. The [AllowSmoothChange](#) property specifies the properties of the layers that support smooth change. The SmoothPropertyEnum type supports the following values.

Name	Value	Description
exSmoothChangeless	0	The change of any of the following properties is not gradually.
exLayerTransparency	1	<a href="#">Transparency</a> , Gets or sets a value that indicates percent of the transparency to display the layer.
exLayerBrightness	2	<a href="#">Brightness</a> , Specifies the percent of brightness to apply to the layer.
exLayerContrast	4	<a href="#">Contrast</a> , Specifies the percent of contrast to apply to the layer.

# Appearance object

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

Name	Description
<a href="#">Add</a>	Adds or replaces a skin object to the control.
<a href="#">Clear</a>	Removes all skins in the control.
<a href="#">Remove</a>	Removes a specific skin from the control.
<a href="#">RenderType</a>	Specifies the way colored EBN objects are displayed on the component.

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

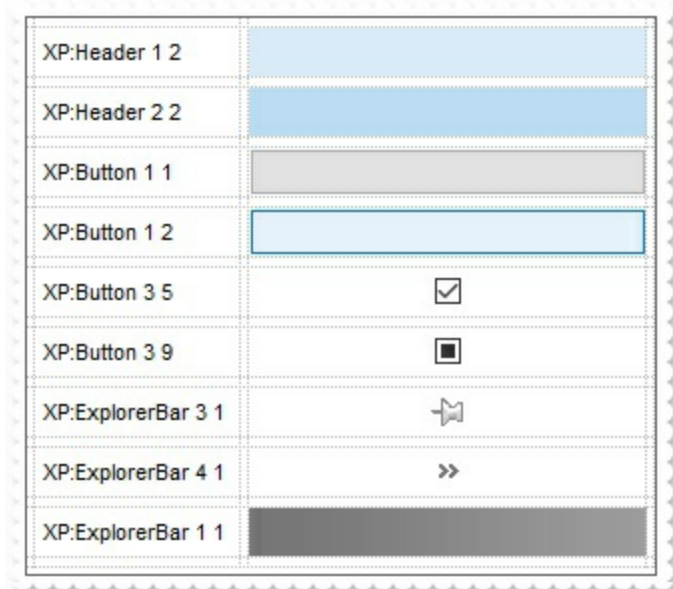
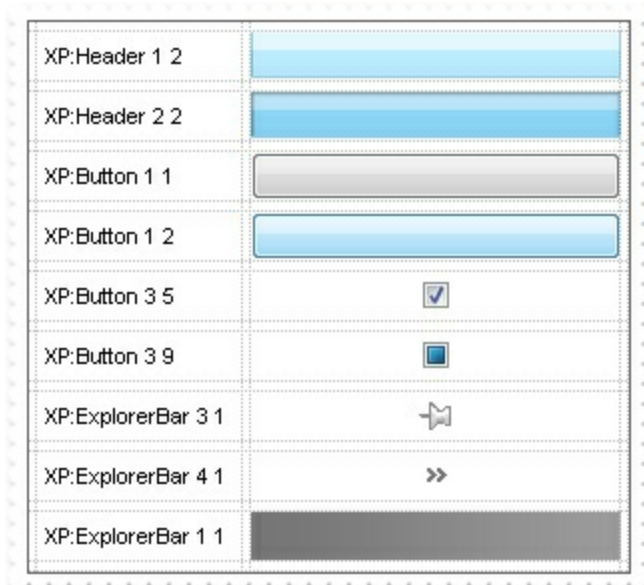
Adds or replaces a skin object to the control.

Type	Description
ID as Long	<p>A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.</p> <p>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 <a href="#">EBN</a> 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) )</p> <p>If the Skin parameter points to a string expression, it can be one of the following:</p> <ul style="list-style-type: none"><li>• A path to the skin file ( *.<a href="#">EBN</a> ). The <a href="#">ExButton</a> component or <a href="#">ExEBN</a> 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"</li><li>• A BASE64 encoded string that holds the skin file ( *.<a href="#">EBN</a> ). Use the <a href="#">ExImages</a> tool to build BASE 64 encoded strings of the skin file ( *.<a href="#">EBN</a> ). The BASE64 encoded string starts with "gBFLBCJw..."</li><li>• 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: "<a href="#">XP:ClassName Part State</a>" 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</li></ul>

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:

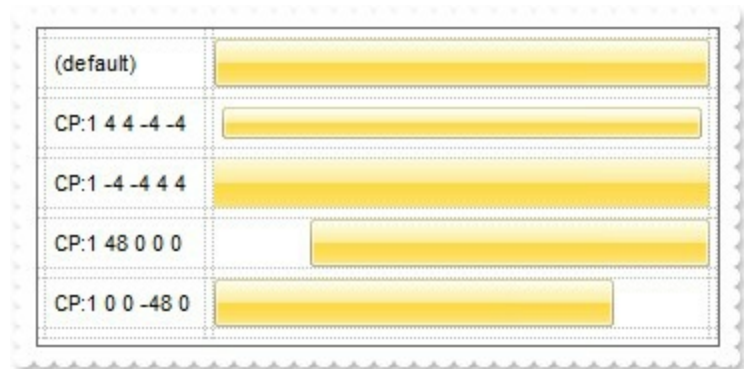
Skin as Variant



- 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

Boolean

## Description

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

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (\*.ebn) assigned to a part of the control. 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 init the control. Use the [Refresh](#) method to refresh the control.

The identifier you choose for the skin is very important to be used in the background properties like explained bellow. Shortly, the color properties uses 4 bytes ( DWORD, double WORD, and so on ) to hold a RGB value. More than that, the first byte ( most

significant byte in the color ) is used only to specify system color. if the first bit in the byte is 1, the rest of bits indicates the index of the system color being used. So, we use the last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. So, since the 7 bits can cover 127 values, excluding 0, we have 126 possibilities to store an identifier in that byte. This way, a DWORD expression indicates the background color stored in RRGGBB format and the index of the skin ( ID parameter ) in the last 7 bits in the high significant byte of the color. For instance, the BackColor = BackColor Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the object that BackColor is applied.

Starting with **Windows XP**, the following table shows how the common controls are broken into parts and states:

Control/ClassName		Part	States
BUTTON	BP_CHECKBOX = 3		CBS_UNCHECKED
			1 CBS_UNCHECKED
			CBS_UNCHECKED
			= 3
			CBS_UNCHECKED
			= 4 CBS_CHECKED
			5 CBS_CHECKED
			CBS_CHECKEDPR
			CBS_CHECKEDDIS
			CBS_MIXEDNORM
			CBS_MIXEDHOT =
			CBS_MIXEDPRES
	BP_GROUPBOX = 4		CBS_MIXEDDISAB
			GBS_NORMAL = 1
			GBS_DISABLED =
			PBS_NORMAL = 1
			= 2 PBS_PRESSED
			PBS_DISABLED =
			PBS_DEFAULTED :
			RBS_UNCHECKED
			1 RBS_UNCHECKED
			RBS_UNCHECKED
			= 3
	BP_RADIOBUTTON = 2		RBS_UNCHECKED
			= 4 RBS_CHECKED
			5 RBS_CHECKED
			RBS_CHECKEDPR
			RBS_CHECKEDDIS

CLOCK	BP_USERBUTTON = 5	CLS_NORMAL = 1
	CLP_TIME = 1	CBXS_NORMAL = 1
COMBOBOX	CP_DROPDOWNBUTTON = 1	CBXS_HOT = 2
		CBXS_PRESSED = 3
EDIT	EP_CARET = 2	CBXS_DISABLED = 4
	EP_EDITTEXT = 1	ETS_NORMAL = 1
EXPLORERBAR	EBP_HEADERBACKGROUND = 1	2 ETS_SELECTED
	EBP_HEADERCLOSE = 2	ETS_DISABLED = 2
	EBP_HEADERPIN = 3	ETS_FOCUSED = 3
	EBP_IEBARMENU = 4	ETS_READONLY = 4
	EBP_NORMALGROUPBACKGROUND = 5	ETS_ASSIST = 7
	EBP_NORMALGROUPCOLLAPSE = 6	EBHC_NORMAL = 1
	EBP_NORMALGROUPEXPAND = 7	EBHC_HOT = 2
	EBP_NORMALGROUPHEAD = 8	EBHC_PRESSED = 3
	EBP_SPECIALGROUPBACKGROUND = 9	EBHP_NORMAL = 1
	EBP_SPECIALGROUPCOLLAPSE = 10	EBHP_HOT = 2
	EBP_SPECIALGROUPEXPAND = 11	EBHP_PRESSED = 3
		EBHP_SELECTED = 4
		4 EBHP_SELECTED
		EBHP_SELECTED = 6
		EBM_NORMAL = 1
		= 2 EBM_PRESSED
		EBNGC_NORMAL = 1
		EBNGC_HOT = 2
		EBNGC_PRESSED = 3
		EBNGE_NORMAL = 1
		EBNGE_HOT = 2
		EBNGE_PRESSED = 3
		EBSGC_NORMAL = 1
		EBSGC_HOT = 2
		EBSGC_PRESSED = 3
		EBSGE_NORMAL = 1
		EBSGE_HOT = 2

	EBP_SPECIALGROUPHEAD = 12	EBSGE_PRESSED = 1
HEADER	HP_HEADERITEM = 1	HIS_NORMAL = 1 2 HIS_PRESSED = 2
	HP_HEADERITEMLEFT = 2	HILS_NORMAL = 1 = 2 HILS_PRESSED = 2
	HP_HEADERITEMRIGHT = 3	HIRS_NORMAL = 1 = 2 HIRS_PRESSED = 2
	HP_HEADERSORTARROW = 4	HSAS_SORTEDUP = 1 HSAS_SORTEDDC = 2
LISTVIEW	LVP_EMPTYTEXT = 5	
	LVP_LISTDETAIL = 3	
	LVP_LISTGROUP = 2	
	LVP_LISTITEM = 1	LIS_NORMAL = 1 2 LIS_SELECTED = 2 LIS_DISABLED = 4 LIS_SELECTEDNO = 5
	LVP_LISTSORTEDDETAIL = 4	
MENU	MP_MENUBARDROPDOWN = 4	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_MENUBARITEM = 3	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_CHEVRON = 5	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_MENUDROPDOWN = 2	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_MENUITEM = 1	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_SEPARATOR = 6	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
MENUBAND	MDP_NEWAPPBUTTON = 1	MDS_NORMAL = 1 = 2 MDS_PRESSED = 2 MDS_DISABLED = 3



		MDS_CHECKED = MDS_HOTCHECKED
	MDP_SEPERATOR = 2	
PAGE	PGRP_DOWN = 2	DNS_NORMAL = 1 = 2 DNS_PRESSED DNS_DISABLED = DNHZS_NORMAL = DNHZS_HOT = 2 DNHZS_PRESSED DNHZS_DISABLED
	PGRP_DOWNHORZ = 4	UPS_NORMAL = 1 = 2 UPS_PRESSED UPS_DISABLED = UPHZS_NORMAL = UPHZS_HOT = 2 UPHZS_PRESSED UPHZS_DISABLED
	PGRP_UP = 1	
	PGRP_UPHORZ = 3	
PROGRESS	PP_BAR = 1 PP_BARVERT = 2 PP_CHUNK = 3 PP_CHUNKVERT = 4	
REBAR	RP_BAND = 3  RP_CHEVRON = 4  RP_CHEVRONVERT = 5 RP_GRIPPER = 1 RP_GRIPPERVERT = 2	CHEVS_NORMAL = CHEVS_HOT = 2 CHEVS_PRESSED
		ABS_DOWNDISAB ABS_DOWNHOT, ABS_DOWNNORM ABS_DOWNPRES ABS_UPDISABLED ABS_UPHOT, ABS_UPNORMAL, ABS_UPPRESSED, ABS_LEFTDISABLI ABS_LEFTHOT, ABS_LEFTNORMA ABS_LEFTPRESSE
SCROLLBAR	SBP_ARROWBTN = 1	

## SPIN

SBP\_GRIPPERHORZ = 8

SBP\_GRIPPERVERT = 9

SBP\_LOWERTRACKHORZ = 4

SBP\_LOWERTRACKVERT = 6

SBP\_THUMBBTNHORZ = 2

SBP\_THUMBBTNVERT = 3

SBP\_UPPERTRACKHORZ = 5

SBP\_UPPERTRACKVERT = 7

SBP\_SIZEBOX = 10

SPNP\_DOWN = 2

SPNP\_DOWNHORZ = 4

SPNP\_UP = 1

ABS\_RIGHTDISAB  
ABS\_RIGHTHOT,  
ABS\_RIGHTNORM  
ABS\_RIGHTPRESS

SCRBS\_NORMAL :  
SCRBS\_HOT = 2  
SCRBS\_PRESSED  
SCRBS\_DISABLED

SCRBS\_NORMAL :  
SCRBS\_HOT = 2  
SCRBS\_PRESSED  
SCRBS\_DISABLED

SCRBS\_NORMAL :  
SCRBS\_HOT = 2  
SCRBS\_PRESSED  
SCRBS\_DISABLED

SCRBS\_NORMAL :  
SCRBS\_HOT = 2  
SCRBS\_PRESSED  
SCRBS\_DISABLED

SCRBS\_NORMAL :  
SCRBS\_HOT = 2  
SCRBS\_PRESSED  
SCRBS\_DISABLED

SCRBS\_NORMAL :  
SCRBS\_HOT = 2  
SCRBS\_PRESSED  
SCRBS\_DISABLED

SZB\_RIGHTALIGN  
SZB\_LEFTALIGN =  
DNS\_NORMAL = 1  
= 2 DNS\_PRESSED

DNS\_DISABLED =  
DNHZZ\_NORMAL :  
DNHZZ\_HOT = 2  
DNHZZ\_PRESSED  
DNHZZ\_DISABLED

UPS\_NORMAL = 1  
= 2 UPS\_PRESSED

SPNP\_UPHORZ = 3

**STARTPANEL**

SPP\_LOGOFF = 8

SPP\_LOGOFFBUTTONS = 9

SPP\_MOREPROGRAMS = 2

SPP\_MOREPROGRAMSARROW = 3

SPP\_PLACESLIST = 6

SPP\_PLACESLISTSEPARATOR = 7

SPP\_PREVIEW = 11

SPP\_PROGLIST = 4

SPP\_PROGLISTSEPARATOR = 5

SPP\_USERPANE = 1

SPP\_USERPICTURE = 10

**STATUS**

SP\_GRIPPER = 3

SP\_PANE = 1

SP\_GRIPPERPANE = 2

**TAB**

TABP\_BODY = 10

TABP\_PANE = 9

TABP\_TABITEM = 1

TABP\_TABITEMBOTHEDGE = 4

TABP\_TABITEMLEFTEDGE = 2

UPS\_DISABLED =  
UPHZS\_NORMAL =  
UPHZS\_HOT = 2  
UPHZS\_PRESSED  
UPHZS\_DISABLED

SPLS\_NORMAL =  
SPLS\_HOT = 2  
SPLS\_PRESSED =

SPS\_NORMAL = 1  
= 2 SPS\_PRESSED

TIS\_NORMAL = 1  
2 TIS\_SELECTED :  
TIS\_DISABLED = 4  
TIS\_FOCUSED = 5  
TIBES\_NORMAL =  
TIBES\_HOT = 2  
TIBES\_SELECTED  
TIBES\_DISABLED  
TIBES\_FOCUSED :  
TILES\_NORMAL =  
TILES\_HOT = 2  
TILES\_SELECTED  
TILES\_DISABLED :  
TILES\_FOCUSED :  
TIRES\_NORMAL =

TABP\_TABITEMRIGHTEDGE = 3

TABP\_TOPTABITEM = 5

TABP\_TOPTABITEMBOTHEDGE = 8

TABP\_TOPTABITEMLEFTEDGE = 6

TABP\_TOPTABITEMRIGHTEDGE = 7

## TASKBAND

TDP\_GROUPCOUNT = 1

TDP\_FLASHBUTTON = 2

TDP\_FLASHBUTTONGROUPMENU = 3

## TASKBAR

TBP\_BACKGROUNDBOTTOM = 1

TBP\_BACKGROUNDLEFT = 4

TBP\_BACKGROUNDRIGHT = 2

TBP\_BACKGROUNDTOP = 3

TBP\_SIZINGBARBOTTOM = 5

TBP\_SIZINGBARBOTTOMLEFT = 8

TBP\_SIZINGBARRIGHT = 6

TBP\_SIZINGBARTOP = 7

## TOOLBAR

TP\_BUTTON = 1

TIRES\_HOT = 2

TIRES\_SELECTED

TIRES\_DISABLED

TIRES\_FOCUSED

TTIS\_NORMAL = 1

= 2 TTIS\_SELECTED

TTIS\_DISABLED =

TTIS\_FOCUSED =

TTIBES\_NORMAL

TTIBES\_HOT = 2

TTIBES\_SELECTED

TTIBES\_DISABLED

TTIBES\_FOCUSED

TTILES\_NORMAL

TTILES\_HOT = 2

TTILES\_SELECTED

TTILES\_DISABLED

TTILES\_FOCUSED

TTIRES\_NORMAL

TTIRES\_HOT = 2

TTIRES\_SELECTED

TTIRES\_DISABLED

TTIRES\_FOCUSED

TS\_NORMAL = 1 T

TS\_PRESSED = 3

TS\_DISABLED = 4

TS\_CHECKED = 5

TS\_HOTCHECKED

TS\_NORMAL = 1 T

TP\_DROPDOWNBUTTON = 2

TP\_SPLITBUTTON = 3

TP\_SPLITBUTTONDROPDOWN = 4

TP\_SEPARATOR = 5

TP\_SEPARATORVERT = 6

## TOOLTIP

TTP\_BALLOON = 3

TTP\_BALLOONTITLE = 4

TTP\_CLOSE = 5

TTP\_STANDARD = 1

TTP\_STANDARDTITLE = 2

## TRACKBAR

TKP\_THUMB = 3

TS\_PRESSED = 3

TS\_DISABLED = 4

TS\_CHECKED = 5

TS\_HOTCHECKED

TS\_NORMAL = 1 T

TS\_PRESSED = 3

TS\_DISABLED = 4

TS\_CHECKED = 5

TS\_HOTCHECKED

TS\_NORMAL = 1 T

TS\_PRESSED = 3

TS\_DISABLED = 4

TS\_CHECKED = 5

TS\_HOTCHECKED

TS\_NORMAL = 1 T

TS\_PRESSED = 3

TS\_DISABLED = 4

TS\_CHECKED = 5

TS\_HOTCHECKED

TS\_NORMAL = 1 T

TS\_PRESSED = 3

TS\_DISABLED = 4

TS\_CHECKED = 5

TS\_HOTCHECKED

TTBS\_NORMAL =

TTBS\_LINK = 2

TTBS\_NORMAL =

TTBS\_LINK = 2

TTCS\_NORMAL =

TTCS\_HOT = 2

TTCS\_PRESSED =

TTSS\_NORMAL =

TTSS\_LINK = 2

TTSS\_NORMAL =

TTSS\_LINK = 2

TUS\_NORMAL = 1

2 TUS\_PRESSED =

TUS\_FOCUSED =

TUS\_DISABLED =

TUBS\_NORMAL =

TUBS\_HOT = 2

TKP\_THUMBBOTTOM = 4

TKP\_THUMBLEFT = 7

TKP\_THUMBRIGHT = 8

TKP\_THUMBTOP = 5

TKP\_THUMBVERT = 6

TKP\_TICS = 9

TKP\_TICSVERT = 10

TKP\_TRACK = 1

TKP\_TRACKVERT = 2

## TRAYNOTIFY

TNP\_ANIMBACKGROUND = 2

TNP\_BACKGROUND = 1

## TREEVIEW

TVP\_BRANCH = 3

TVP\_GLYPH = 2

TVP\_TREEITEM = 1

## WINDOW

WP\_CAPTION = 1

TUBS\_PRESSED =

TUBS\_FOCUSED =

TUBS\_DISABLED =

TUVLS\_NORMAL =

TUVLS\_HOT = 2

TUVLS\_PRESSED

TUVLS\_FOCUSED

TUVLS\_DISABLED

TUVRNORMAL =

TUVRNORMAL\_HOT = 2

TUVRNORMAL\_PRESSED

TUVRNORMAL\_FOCUSED

TUVRNORMAL\_DISABLED

TUTS\_NORMAL =

TUTS\_HOT = 2

TUTS\_PRESSED =

TUTS\_FOCUSED =

TUTS\_DISABLED =

TUVS\_NORMAL =

TUVS\_HOT = 2

TUVS\_PRESSED =

TUVS\_FOCUSED =

TUVS\_DISABLED =

TSS\_NORMAL = 1

TSVS\_NORMAL =

TRS\_NORMAL = 1

TRVS\_NORMAL =

GLPS\_CLOSED =

GLPS\_OPENED =

TREIS\_NORMAL =

TREIS\_HOT = 2

TREIS\_SELECTED

TREIS\_DISABLED

TREIS\_SELECTED

= 5

CS\_ACTIVE = 1 CS

= 2 CS\_DISABLED

WP\_CAPTIONSIZINGTEMPLATE = 30

WP\_CLOSEBUTTON = 18

WP\_DIALOG = 29

WP\_FRAMEBOTTOM = 9

WP\_FRAMEBOTTOMSIZINGTEMPLATE = 36

WP\_FRAMELEFT = 7

WP\_FRAMELEFTSIZINGTEMPLATE = 32

WP\_FRAMERIGHT = 8

WP\_FRAMERIGHTSIZINGTEMPLATE = 34

WP\_HELPBUTTON = 23

WP\_HORIZSCROLL = 25

WP\_HORIZTHUMB = 26

WP\_MAX\_BUTTON

WP\_MAXCAPTION = 5

WP\_MDICLOSEBUTTON = 20

WP\_MDIHELPBUTTON = 24

WP\_MDIMINBUTTON = 16

CBS\_NORMAL = 1  
= 2 CBS\_PUSHED  
CBS\_DISABLED =

FS\_ACTIVE = 1 FS  
= 2

FS\_ACTIVE = 1 FS  
= 2

FS\_ACTIVE = 1 FS  
= 2

HBS\_NORMAL = 1  
= 2 HBS\_PUSHED  
HBS\_DISABLED =

HSS\_NORMAL = 1  
= 2 HSS\_PUSHED  
HSS\_DISABLED =

HTS\_NORMAL = 1  
2 HTS\_PUSHED =  
HTS\_DISABLED =

MAXBS\_NORMAL  
MAXBS\_HOT = 2  
MAXBS\_PUSHED =  
MAXBS\_DISABLED

MXCS\_ACTIVE = 1  
MXCS\_INACTIVE =  
MXCS\_DISABLED

CBS\_NORMAL = 1  
= 2 CBS\_PUSHED  
CBS\_DISABLED =

HBS\_NORMAL = 1  
= 2 HBS\_PUSHED  
HBS\_DISABLED =

MINBS\_NORMAL =  
MINBS\_HOT = 2  
MINBS\_PUSHED =  
MINBS\_DISABLED

WP\_MDIRESTOREBUTTON = 22

WP\_MDISYSBUTTON = 14

WP\_MINBUTTON = 15

WP\_MINCAPTION = 3

WP\_RESTOREBUTTON = 21

WP\_SMALLCAPTION = 2

WP\_SMALLCAPTIONSIZINGTEMPLATE = 31

WP\_SMALLCLOSEBUTTON = 19

WP\_SMALLFRAMEBOTTOM = 12

WP\_SMALLFRAMEBOTTOMSIZINGTEMPLATE  
= 37

WP\_SMALLFRAMELEFT = 10

WP\_SMALLFRAMELEFTSIZINGTEMPLATE =  
33

WP\_SMALLFRAMERIGHT = 11

WP\_SMALLFRAMERIGHTSIZINGTEMPLATE =  
35

WP\_SMALLHELPBUTTON

WP\_SMALLMAXBUTTON

RBS\_NORMAL = 1  
= 2 RBS\_PUSHED  
RBS\_DISABLED =  
SBS\_NORMAL = 1  
= 2 SBS\_PUSHED  
SBS\_DISABLED =  
MINBS\_NORMAL =  
MINBS\_HOT = 2  
MINBS\_PUSHED =  
MINBS\_DISABLED  
MNCS\_ACTIVE = 1  
MNCS\_INACTIVE =  
MNCS\_DISABLED  
RBS\_NORMAL = 1  
= 2 RBS\_PUSHED  
RBS\_DISABLED =  
CS\_ACTIVE = 1 CS  
= 2 CS\_DISABLED

CBS\_NORMAL = 1  
= 2 CBS\_PUSHED  
CBS\_DISABLED =  
FS\_ACTIVE = 1 FS  
= 2

FS\_ACTIVE = 1 FS  
= 2

FS\_ACTIVE = 1 FS  
= 2

HBS\_NORMAL = 1  
= 2 HBS\_PUSHED  
HBS\_DISABLED =  
MAXBS\_NORMAL  
MAXBS\_HOT = 2  
MAXBS\_PUSHED =



WP\_SMALLMAXCAPTION = 6

WP\_SMALLMINCAPTION = 4

WP\_SMALLRESTOREBUTTON

WP\_SMALLSYSBUTTON

WP\_SYSBUTTON = 13

WP\_VERTSCROLL = 27

WP\_VERTTHUMB = 28

MAXBS\_DISABLED = 0  
MXCS\_ACTIVE = 1  
MXCS\_INACTIVE = 0  
MXCS\_DISABLED = 0  
MNCS\_ACTIVE = 1  
MNCS\_INACTIVE = 0  
MNCS\_DISABLED = 0  
RBS\_NORMAL = 1  
RBS\_PUSHED = 2  
RBS\_DISABLED = 0

SBS\_NORMAL = 1  
SBS\_PUSHED = 2  
SBS\_DISABLED = 0  
SBS\_NORMAL = 1  
SBS\_PUSHED = 2  
SBS\_DISABLED = 0  
VSS\_NORMAL = 1  
VSS\_PUSHED = 2  
VSS\_DISABLED = 0  
VTS\_NORMAL = 1  
VTS\_PUSHED = 2  
VTS\_DISABLED = 0

# method Appearance.Clear ()

Removes all skins in the control.

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

Use the Clear method to clear all skins from the control. Use the [Remove](#) method to remove a specific skin. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

# method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

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

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


# 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.

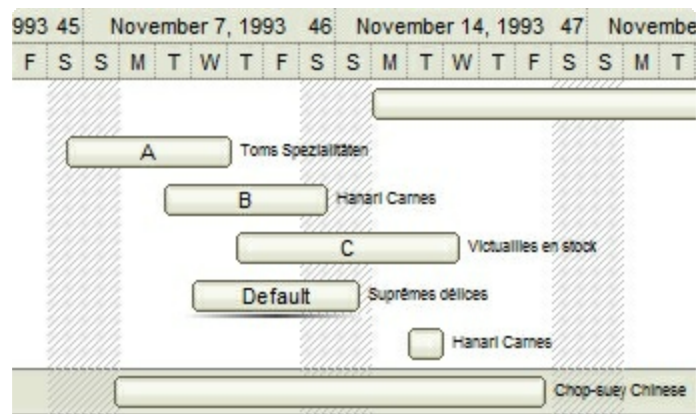
Click here  to watch a movie on how you can change the colors to be applied on EBN objects.

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

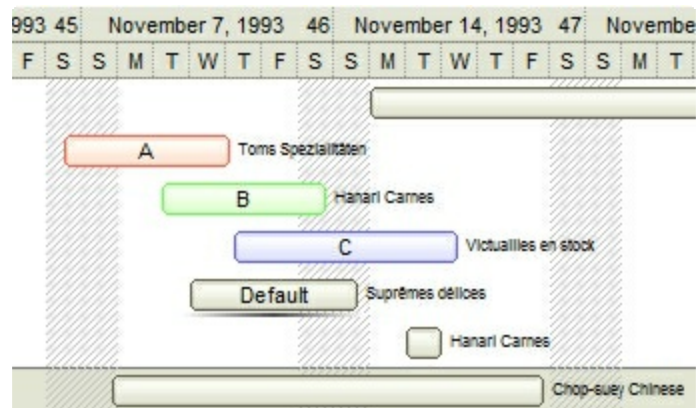
- "A" in Red ( RGB(255,0,0 ) , for instance the bar's property exBarColor is 0x10000FF
- "B" in Green ( RGB(0,255,0 ) , for instance the bar's property exBarColor is 0x100FF00
- "C" in Blue ( RGB(0,0,255 ) , for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

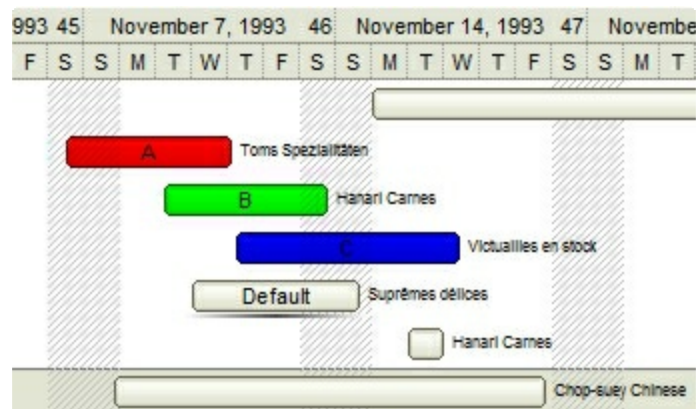
- **-3**, *no color is applied*. For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color ( Blue color ) is ignored. You can use this option to allow the control displays the EBN colors or not.



- **-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), ... )

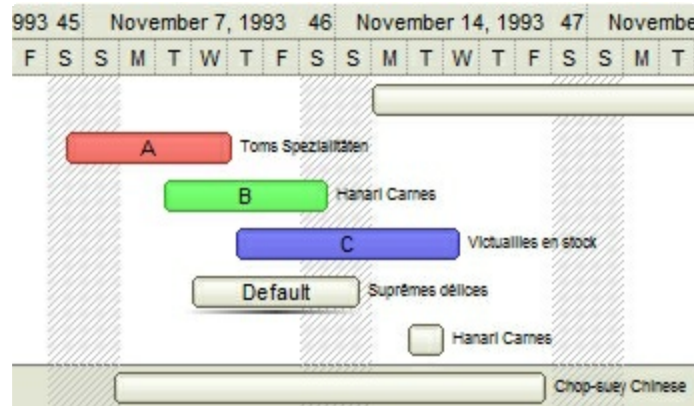


- **-1, AND-color scheme,** The color to be applied on the part of the control is an AND bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the AND bit for the entire Blue channel, or in other words, it applies a more Blue to 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), ... )



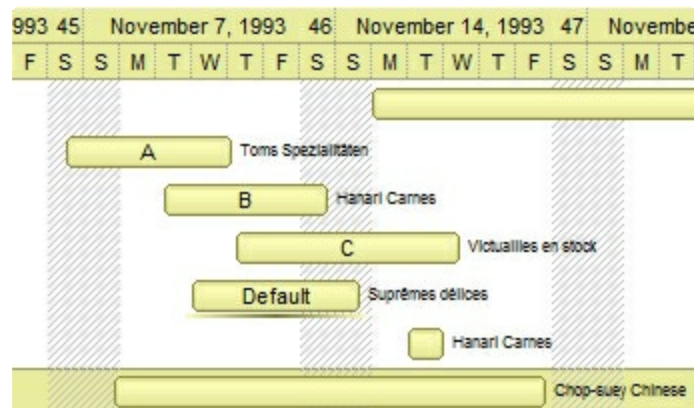
- **0, default,** the specified color is applied to the EBN. For instance, the

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.

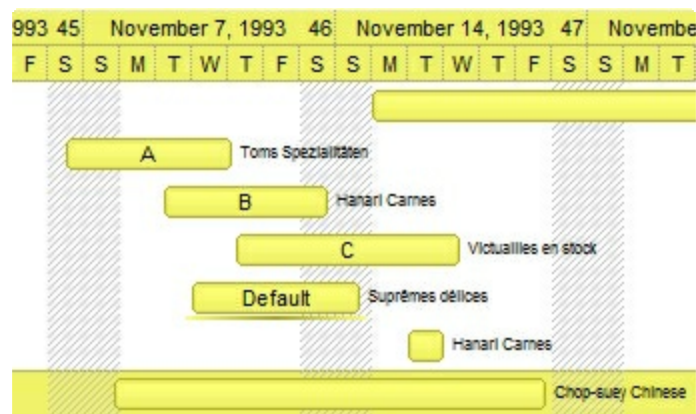


- **0xAABBGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on 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.

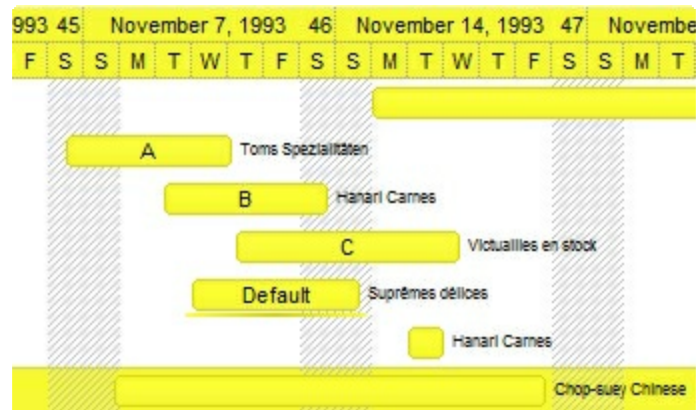
*The following picture shows the control with the RenderType property on 0x4000FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256 ):*



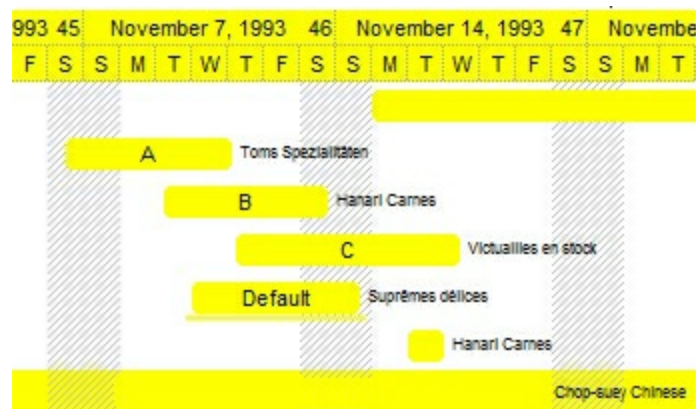
*The following picture shows the control with the RenderType property on 0x8000FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256 ):*



The following picture shows the control with the *RenderType* property on `0xC000FFFF` (75% Yellow, `0xC0` or 192 in decimal is 75% from 256 ):



The following picture shows the control with the *RenderType* property on `0xFF00FFFF` (100% Yellow, `0xFF` or 255 in decimal is 100% from 255 ):

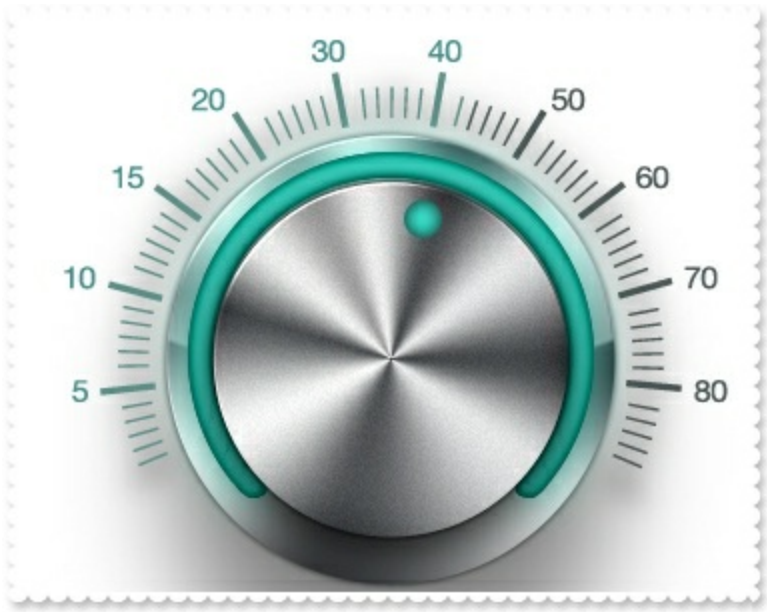




# Background object

The Background object holds pictures to be shown on the layer's background. The [Foreground](#) object holds the HTML captions to be shown on the layer's foreground. The [Background](#) property of the Layer access the layer's Background object. The layer's background can be visible or selectable. If not selectable, the user can not select it runtime, such as LayerFromPoint property ignores it. The Layer's background can display unlimited graphics of different sizes and positions.

The following screen shot shows a pictures on each layer's background:



The Background object supports the following properties and methods:

Name	Description
<a href="#">Color</a>	Indicates the layer's Color object, so you can apply a solid color on the layer's background.
<a href="#">ExtraPicture</a>	Indicates the layer's extra Picture object, so you can show any graphic on the layer's background.
<a href="#">Picture</a>	Indicates the layer's Picture object, so you can show any graphic on the layer's background.
<a href="#">Selectable</a>	Returns or sets a value that indicates whether all objects on the layer's background are selectable.
<a href="#">Visible</a>	Specifies if the objects of the layer's background are shown or hidden.



## property Background.Color as LColor

Indicates the layer's Color object, so you can apply a solid color on the layer's background.

Type	Description
LColor	A <a href="#">LColor</a> object that holds the solid / EBN color to be applied on the layer's background.

By default, the layer's background is transparent. The [Picture](#) / [ExtraPicture](#) property should be used to place a picture on the layer's background. Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background.

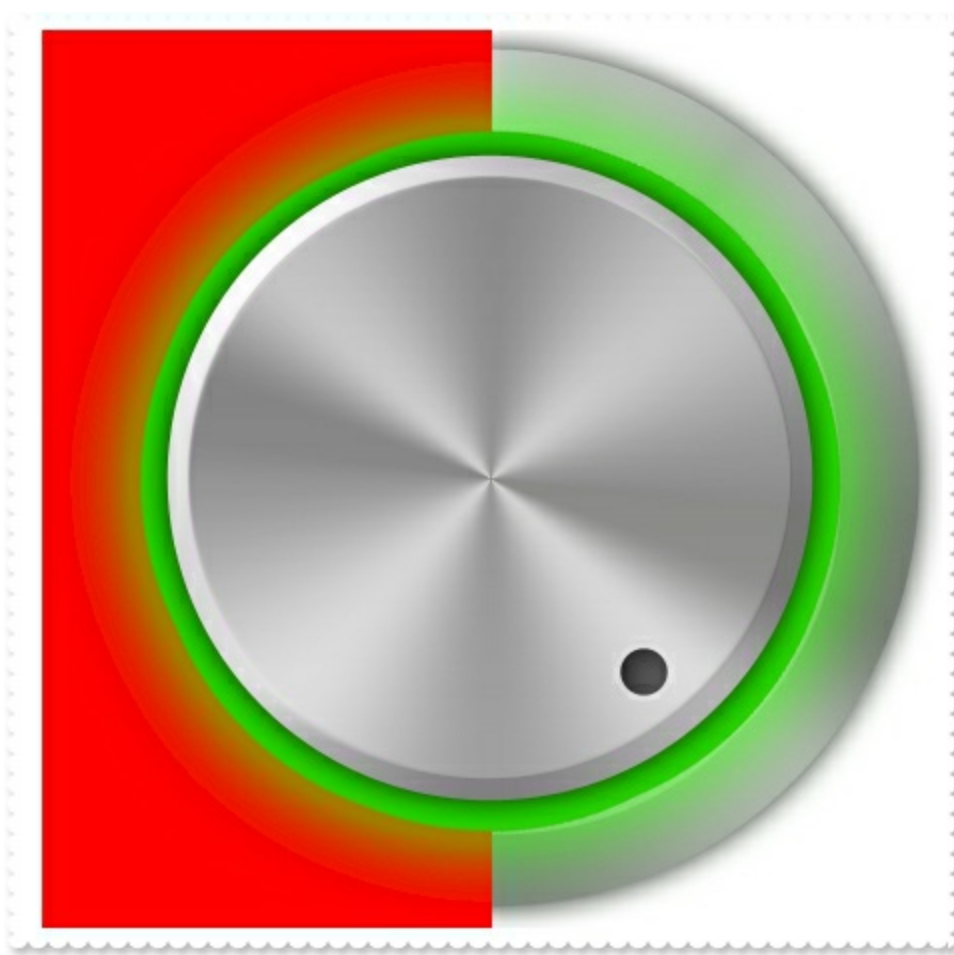
The following properties can be used to move / resize the layer:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

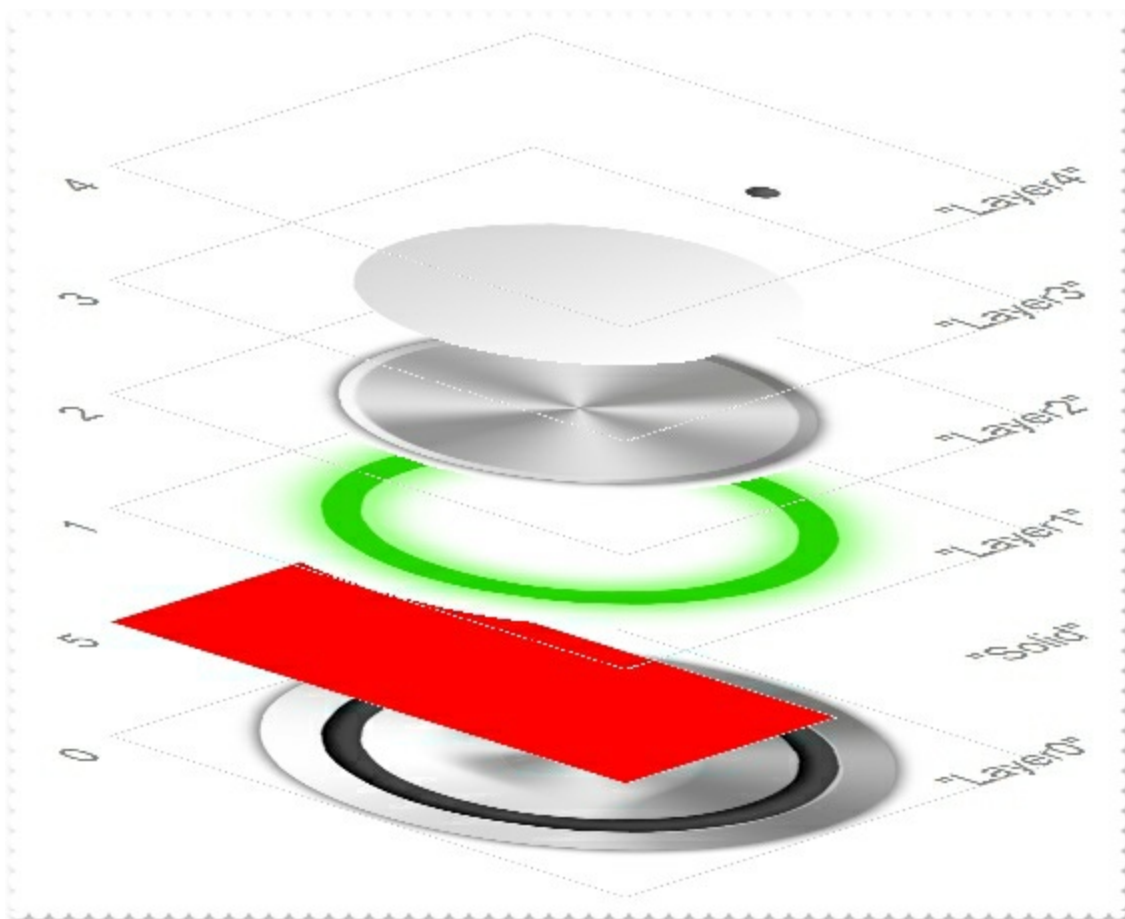
You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.

The following screen shot shows a layer with solid red color:



And if we decompose the layers we get:



The following samples show how you can apply a solid color to be display on left-half of the

layer after the first visible layer:

## VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 5
    With .Layers.Add("Solid")
        .Position = 1
        .Width = "width/2"
        .Background.Color.Value = RGB(255,0,0)
    End With
    .EndUpdate
End With
```

## VB6

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 5
    With .Layers.Add("Solid")
        .Position = 1
        .Width = "width/2"
        .Background.Color.Value = RGB(255,0,0)
    End With
    .EndUpdate
End With
```

## VB.NET

```
With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
```

```

1"
.PicturesName = ""Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
With .Layers.Add("Solid")
    .Position = 1
    .Width = "width/2"
    .Background.Color.Value = Color.FromArgb(255,0,0)
End With
.EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
    .PicturesName = ""Layer` + int(value + 1) + `.png`"
    .Layers.Count = 5
    With .Layers.Add("Solid")
        .Position = 1
        .Width = "width/2"
        .Background.Color.Value = RGB(255,0,0)
    End With
    .EndUpdate()
End With

```

## C++

```

/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
    Library'

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-

```

```

> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(5);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->Add("Solid");
    var_Layer->PutPosition(1);
    var_Layer->PutWidth(L"width/2");
    var_Layer->GetBackground()->GetColor()->PutValue(RGB(255,0,0));
spGauge1->EndUpdate();

```

## C++ Builder

```

Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`";
Gauge1->Layers->Count = 5;
Exgauge1lib_tlb::ILayerPtr var_Layer = Gauge1->Layers->Add(TVariant("Solid"));
    var_Layer->Position = 1;
    var_Layer->Width = L"width/2";
    var_Layer->Background->Color->Value = RGB(255,0,0);
Gauge1->EndUpdate();

```

## C#

```

exgauge1.BeginUpdate();
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
exgauge1.PicturesName = "Layer` + int(value + 1) + `.png`;
exgauge1.Layers.Count = 5;
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers.Add("Solid");
    var_Layer.Position = 1;
    var_Layer.Width = "width/2";
    var_Layer.Background.Color.Value = Color.FromArgb(255,0,0);

```

```
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    Gauge1.Layers.Count = 5;
    var var_Layer = Gauge1.Layers.Add("Solid");
    var_Layer.Position = 1;
    var_Layer.Width = "width/2";
    var_Layer.Background.Color.Value = 255;
    Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>
```

## VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .PicturesPath = "C:\\Program
```

```

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
With .Layers.Add("Solid")
    .Position = 1
    .Width = "width/2"
    .Background.Color.Value = RGB(255,0,0)
End With
.EndUpdate
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 5;
EXGAUGELib.Layer var_Layer = axGauge1.Layers.Add("Solid");
    var_Layer.Position = 1;
    var_Layer.Width = "width/2";
    var_Layer.Background.Color.Value =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Background,com_Color,com_Layer;
    anytype var_Background,var_Color,var_Layer;
    ;

```

```

super();

exgauge1.BeginUpdate();
exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
exgauge1.Layers().Count(5);
var_Layer = COM::createFromObject(exgauge1.Layers()).Add("Solid"); com_Layer =
var_Layer;
    com_Layer.Position(1);
    com_Layer.Width("width/2");
    var_Background = COM::createFromObject(com_Layer.Background());
com_Background = var_Background;
    var_Color = COM::createFromObject(com_Background).Color(); com_Color =
var_Color;
    com_Color.Value(WinApi::RGB2int(255,0,0));
exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
    PicturesName := `Layer` + int(value + 1) + `.png`;
    Layers.Count := 5;
    with Layers.Add('Solid') do
    begin
        Position := 1;
        Width := 'width/2';
        Background.Color.Value := $ff;
    end;
    EndUpdate();
end

```

## Delphi (standard)



```

with Gauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
  PicturesName := '\Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  with Layers.Add('Solid') do
  begin
    Position := 1;
    Width := 'width/2';
    Background.Color.Value := $ff;
  end;
  EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "\Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
with .Layers.Add("Solid")
.Position = 1
.Width = "width/2"
.Background.Color.Value = RGB(255,0,0)
endwith
.EndUpdate
endwith

```

## dBASE Plus

```

local oGauge,var_Layer

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()

```

```

oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
var_Layer = oGauge.Layers.Add("Solid")
    var_Layer.Position = 1
    var_Layer.Width = "width/2"
    var_Layer.Background.Color.Value = 0xff
oGauge.EndUpdate()

```

## XBasic (Alpha Five)

```

Dim oGauge as P
Dim var_Layer as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
var_Layer = oGauge.Layers.Add("Solid")
    var_Layer.Position = 1
    var_Layer.Width = "width/2"
    var_Layer.Background.Color.Value = 255
oGauge.EndUpdate()

```

## Visual Objects

```

local var_Layer as ILayer

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 5

```

```
var_Layer := oDCOCX_Exontrol1:Layers:Add("Solid")
var_Layer:Position := 1
var_Layer:Width := "width/2"
var_Layer:Background:Color:Value := RGB(255,0,0)
oDCOCX_Exontrol1:EndUpdate()
```

## PowerBuilder

```
OleObject oGauge,var_Layer

oGauge = ole_1.Object
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
var_Layer = oGauge.Layers.Add("Solid")
var_Layer.Position = 1
var_Layer.Width = "width/2"
var_Layer.Background.Color.Value = RGB(255,0,0)
oGauge.EndUpdate()
```

## Visual DataFlex

```
Procedure OnCreate
Forward Send OnCreate
Send ComBeginUpdate
Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers
Get ComLayers to voLayers
Handle hoLayers
Get Create (RefClass(cComLayers)) to hoLayers
Set pvComObject of hoLayers to voLayers
Set ComCount of hoLayers to 5
```

```
Send Destroy to hoLayers
Variant voLayers1
Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer
    Get ComAdd of hoLayers1 "Solid" to voLayer
    Handle hoLayer
    Get Create (RefClass(cComLayer)) to hoLayer
    Set pvComObject of hoLayer to voLayer
        Set ComPosition of hoLayer to 1
        Set ComWidth of hoLayer to "width/2"
        Variant voBackground
        Get ComBackground of hoLayer to voBackground
        Handle hoBackground
        Get Create (RefClass(cComBackground)) to hoBackground
        Set pvComObject of hoBackground to voBackground
            Variant voColor
            Get ComColor of hoBackground to voColor
            Handle hoColor
            Get Create (RefClass(cComColor)) to hoColor
            Set pvComObject of hoColor to voColor
                Set ComValue of hoColor to (RGB(255,0,0))
            Send Destroy to hoColor
        Send Destroy to hoBackground
    Send Destroy to hoLayer
Send Destroy to hoLayers1
Send ComEndUpdate
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oGauge

LOCAL oLayer

oForm := XbpDialog():new( AppDesktop() )

oForm:drawingArea:clipChildren := .T.

oForm:create( „{100,100}, {640,480}„, .F. )

oForm:close := {|| PostAppEvent( xbeP\_Quit )}

oGauge := XbpActiveXControl():new( oForm:drawingArea )

oGauge:CLSID := "Exontrol.Gauge.1" /\*{91628F12-393C-44EF-A558-83ED1790AAD3}\*/

oGauge:create(„ {10,60},{610,370} )

oGauge:BeginUpdate()

oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"

oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"

oGauge:Layers():Count := 5

oLayer := oGauge:Layers():Add("Solid")

oLayer:Position := 1

oLayer:Width := "width/2"

oLayer:Background():Color():SetProperty("Value",AutomationTranslateColor(  
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))

oGauge:EndUpdate()

oForm:Show()

DO WHILE nEvent != xbeP\_Quit

nEvent := AppEvent( @mp1, @mp2, @oXbp )

oXbp:handleEvent( nEvent, mp1, mp2 )

ENDDO

RETURN

## property Background.ExtraPicture (Key as Variant) as LPicture

Indicates the layer's extra Picture object, so you can show any graphic on the layer's background.

Type	Description
Key as Variant	Any VARIANT expression that identify the extra-picture to be shown on the layer's background.
LPicture	A <a href="#">LPicture</a> object that specifies the extra-picture to be shown on the layer's background.

The Layer's background can display unlimited graphics of different sizes and positions. The [Picture](#) / ExtraPicture property adds a picture on the layer's background. Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background.

The following properties can be used to move / resize the picture on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following properties can be used to load / import ( manually or automatically ) pictures to the layer's background:

- [PicturesPath](#) property, specifies the path to load pictures from.
- [PicturesName](#) property, specifies the expression that defines the name of the file from the PicturesPath folder to be loaded.
- [Picture.Name](#) / [Picture.Value](#) property of the Background.Picture object, defines the name of the file to be loaded ( relative, absolute, encoded or Picture object )

The [PicturesPath](#) / [PicturesName](#) properties can be used to automatically loads files from a specified folder to be displayed on the layer's background.

For instance,

```
PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob",
```

defines default folder to load pictures from.

`PicturesName = ""Layer` + str(value + 1) + `.png`"`, defines the name of the picture file to be loaded by the layer with the index / value. It defines the names as: Layer1.png for the layer with the index 0, Layer2.png for the layer with the index 1, Layer3.png for the layer with the index 2, and so on.

The [Picture.Name](#) / [Picture.Value](#) property of the Picture object loads a picture / graphics to be displayed on the layer's background.

The Name / Value property could be one of the following:

- A String expression indicates:
  - a name of a picture file in the PicturePath folder. For instance, Name = "Layer1.png", loads the Layer1.png file if found in the PicturePath folder.
  - a picture file including its absolute path. For instance, Name = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob\Layer1.png", loads the Layer1.png file from absolute path
  - a key of the HTML picture, previously loaded by the HTMLPicture method. For instance, Name = "pic1", loads the HTML picture with the key pic1, so the pic1 should be load previously with a HTMLPicture call like `HTMLPicture("pic1") = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob\Layer1.png"`
  - an encode BASE64 string of a picture file. The Exontrol's [ExImages](#) Tool encode/decode BASE64 strings from/to pictures. In this case, the string starts with "gB..", "gC.." and so on.
- A Picture object that indicates the picture to be displayed. For instance, Name = `LoadPicture("picture.jpg")`

# property Background.Picture as LPicture

Indicates the layer's Picture object, so you can show any graphic on the layer's background.

Type	Description
LPicture	A <a href="#">LPicture</a> object that specifies the picture to be shown on the layer's background.

The Layer's background can display unlimited graphics of different sizes and positions. The Picture / [ExtraPicture](#) property adds a picture on the layer's background. Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background.

The following properties can be used to move / resize the picture on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following properties can be used to load / import ( manually or automatically ) pictures to the layer's background:

- [PicturesPath](#) property, specifies the path to load pictures from.
- [PicturesName](#) property, specifies the expression that defines the name of the file from the PicturesPath folder to be loaded.
- [Picture.Name](#) / [Picture.Value](#) property of the Background.Picture object, defines the name of the file to be loaded ( relative, absolute, encoded or Picture object )

The [PicturesPath](#) / [PicturesName](#) properties can be used to automatically loads files from a specified folder to be displayed on the layer's background.

For instance,

PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob",  
defines default folder to load pictures from.

PicturesName = ""Layer` + str(value + 1) + `.png`", defines the name of the picture file



to be loaded by the layer with the index / value. It defines the names as: Layer1.png for the layer with the index 0, Layer2.png for the layer with the index 1, Layer3.png for the layer with the index 2, and so on.

The [Picture.Name](#) / [Picture.Value](#) property of the Picture object loads a picture / graphics to be displayed on the layer's background.

The Name / Value property could be one of the following:

- A String expression indicates:
  - a name of a picture file in the PicturePath folder. For instance, Name = "Layer1.png", loads the Layer1.png file if found in the PicturePath folder.
  - a picture file including its absolute path. For instance, Name = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob\Layer1.png", loads the Layer1.png file from absolute path
  - a key of the HTML picture, previously loaded by the HTMLPicture method. For instance, Name = "pic1", loads the HTML picture with the key pic1, so the pic1 should be load previously with a HTMLPicture call like HTMLPicture("pic1") = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob\Layer1.png"
  - an encode BASE64 string of a picture file. The Exontrol's [ExImages](#) Tool encode/decode BASE64 strings from/to pictures. In this case, the string starts with "gB..", "gC.." and so on.
- A Picture object that indicates the picture to be displayed. For instance, Name = LoadPicture("picture.jpg")

## property Background.Selectable as Boolean

Returns or sets a value that indicates whether all objects on the layer's background are selectable.

Type	Description
Boolean	A Boolean expression that specifies whether the entire layer's background is selectable.

By default, the Selectable property is True, so the user can select the layer if the cursor hovers it. The Selectable property specifies whether the layer's background is selectable. You can use the [Grayscale](#) property to show the entire layer in gray scale ( disable state). The [Visible](#) property specifies whether the layer's background is visible or hidden. The [Picture](#) / [ExtraPicture](#) property adds a picture on the layer's background. Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background.

The Selectable property of the Background object, affects all the pictures / colors being shown on the layer's background. In order to prevent selecting portions of the layer you can use any of the following properties:

- [Selectable](#) property of the LPicture object, returns or sets a value that indicates whether the picture is selectable.
- [Selectable](#) property of the LColor object, returns or sets a value that indicates whether the color is selectable.

# property Background.Visible as Boolean

Specifies if the objects of the layer's background are shown or hidden.

Type	Description
Boolean	A Boolean expression that specifies whether the entire layer's background is visible or hidden.

By default, the Visible property is True, so any picture on the layer's background is visible. The Visible property specifies whether the layer's background is visible or hidden. The [Picture](#) / [ExtraPicture](#) property adds a picture on the layer's background. Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background. The [Selectable](#) property specifies whether the layer's background is selectable.

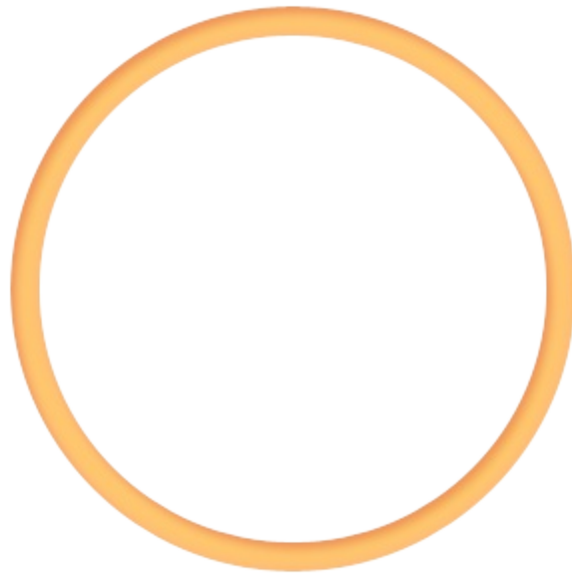
The Visible property of the Background object, affects all the pictures / colors being shown on the layer's background. In order to prevent showing portions of the layer you can use any of the following properties:

- [Visible](#) property of the LPicture object, specifies if the picture is shown or hidden on the layer's background.
- [Visible](#) property of the LColor object, specifies if the color is visible or hidden.

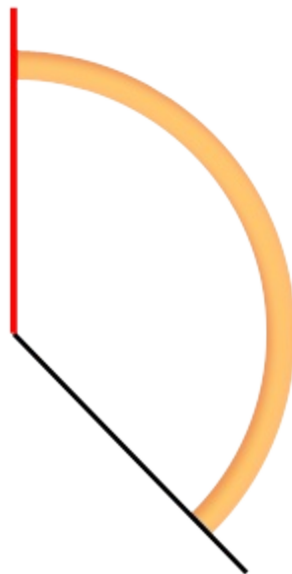
# Clip object

The Clip object defines the clipping you can apply to any layer on the control. The Clipping support include intersection of any of rectangle, round rectangle, ellipse, pie, picture mask, polygon, and so on. The [Clip](#) property accesses the layer's Clip object.

Having the following layer:



By clipping, we can get something like follows:



and if we display the entire gauge here's what we get:



The Clip object supports the following properties and methods:

Name	Description
<a href="#">Ellipse</a>	Gets access to the layer's ellipse clip object.
<a href="#">Picture</a>	Gets access to the layer's picture clip object.
<a href="#">Pie</a>	Gets access to the layer's pie clip object.
<a href="#">Polygon</a>	Gets access to the layer's polygon clip object.
<a href="#">Rectangle</a>	Gets access to the layer's rectangle clip object.
<a href="#">RoundRectangle</a>	Gets access to the layer's round rectangle clip object.
<a href="#">Type</a>	Specifies the type of the clipping the current layer supports.
<a href="#">Value</a>	Indicates the object's value.

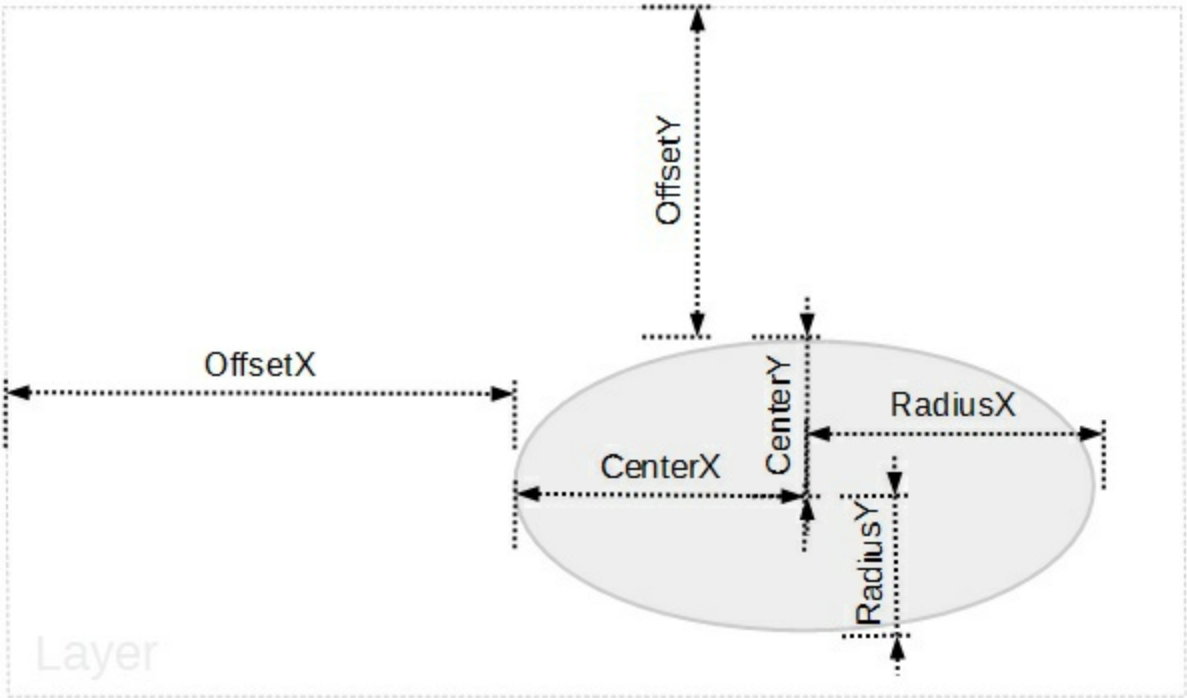
# property Clip.Ellipse as ClipEllipse

Gets access to the layer's ellipse clip object.

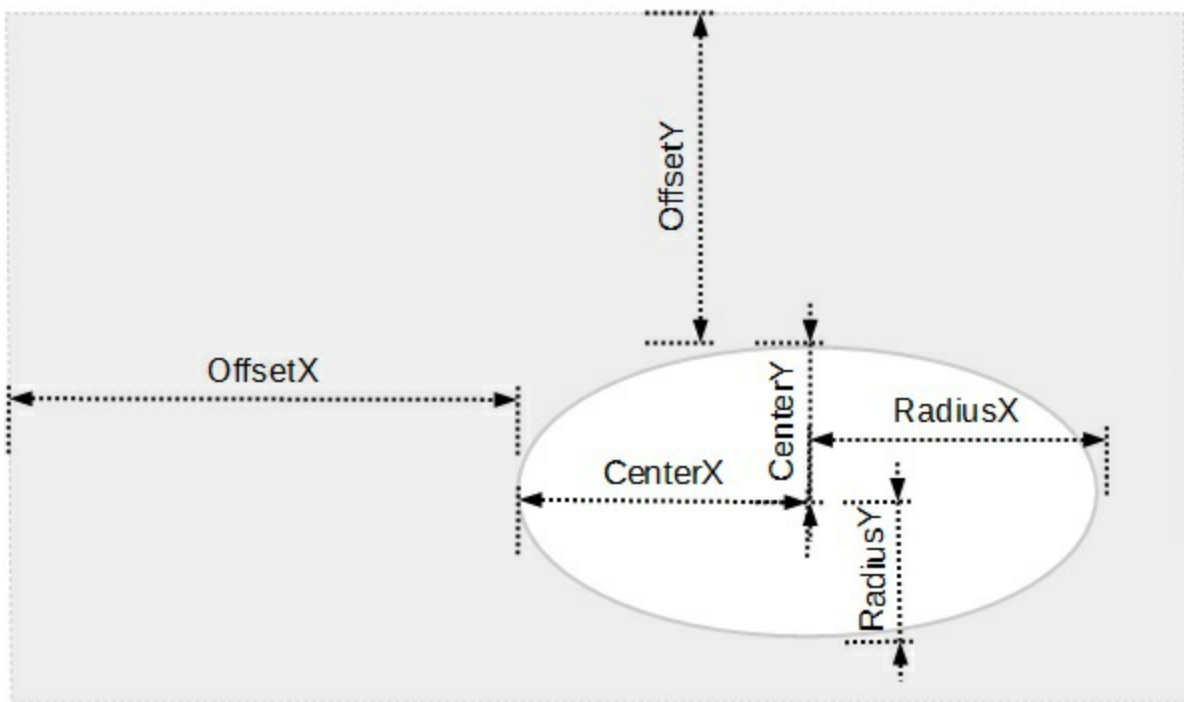
Type	Description
ClipEllipse	A <a href="#">ClipEllipse</a> object that holds information about elliptical clip region

The Ellipse property gets access to the layer's ellipse clip object.

The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is False, by default):



The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is True):



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

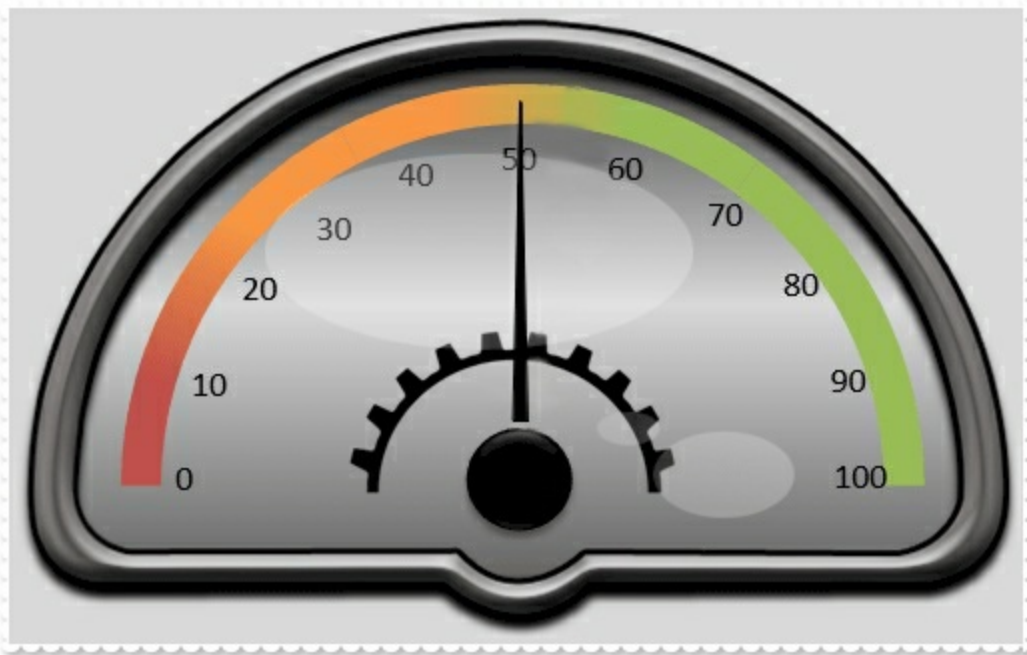
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

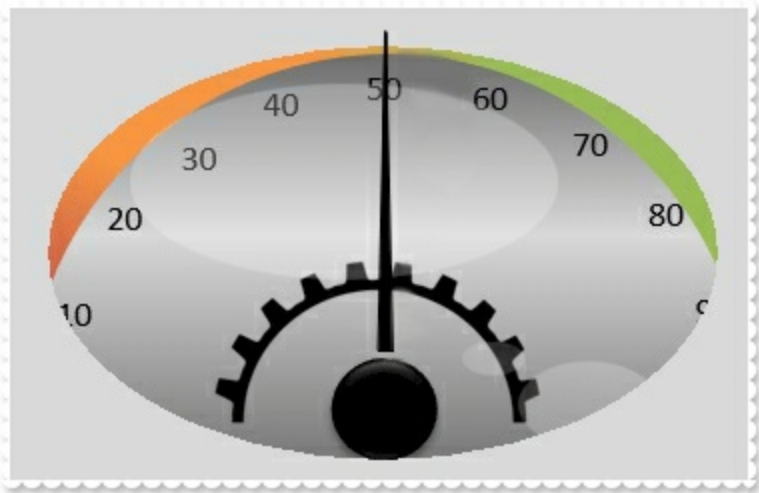
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

For instance, having the following gauge:



an elliptical clip region over the background layer shows as:





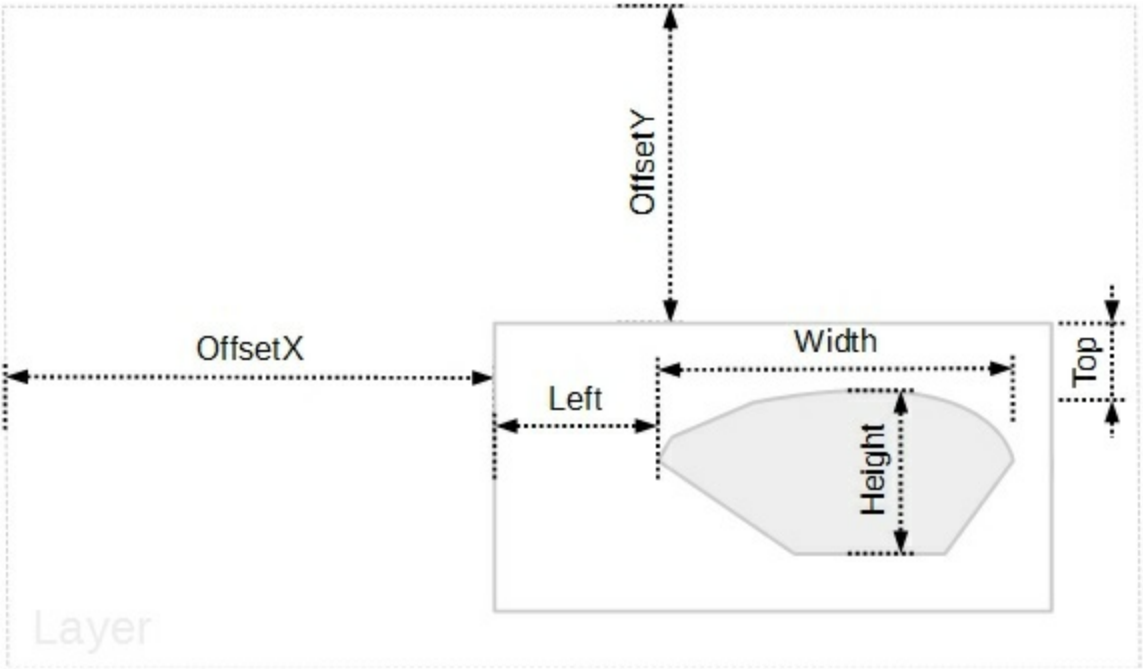
# property Clip.Picture as ClipPicture

Gets access to the layer's picture clip object.

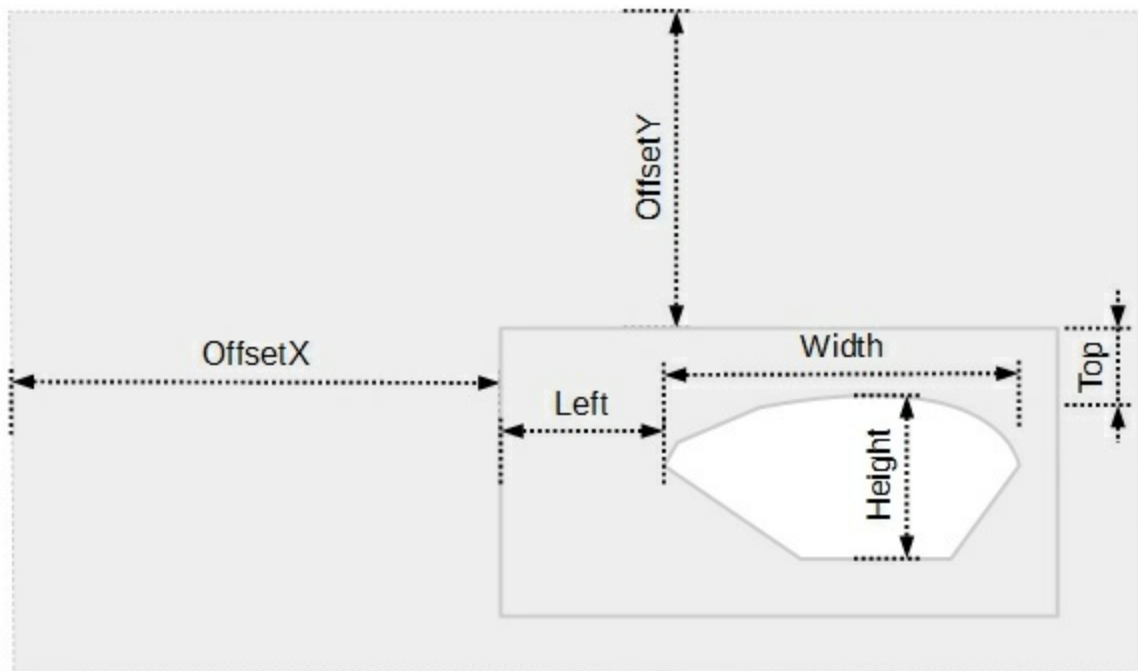
Type	Description
ClipPicture	A <a href="#">ClipPicture</a> object that holds information about picture clip region

The Picture property gets access to the layer's picture clip object.

The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is False, by default):



The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is True):



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

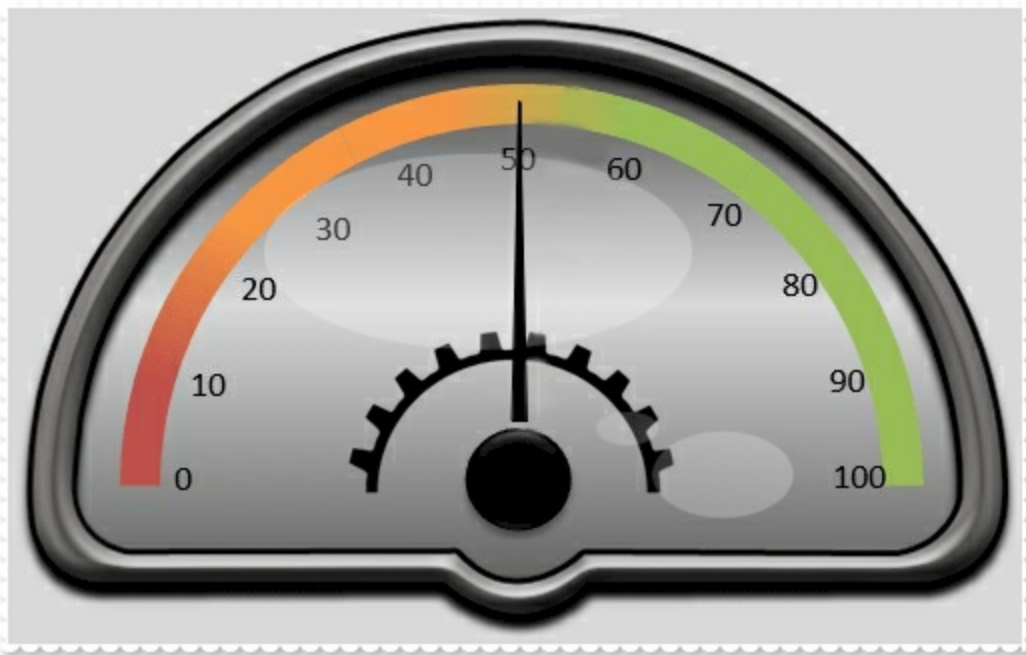
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

For instance, having the following gauge:



a picture clip region over the background layer shows as:



when the source picture is:



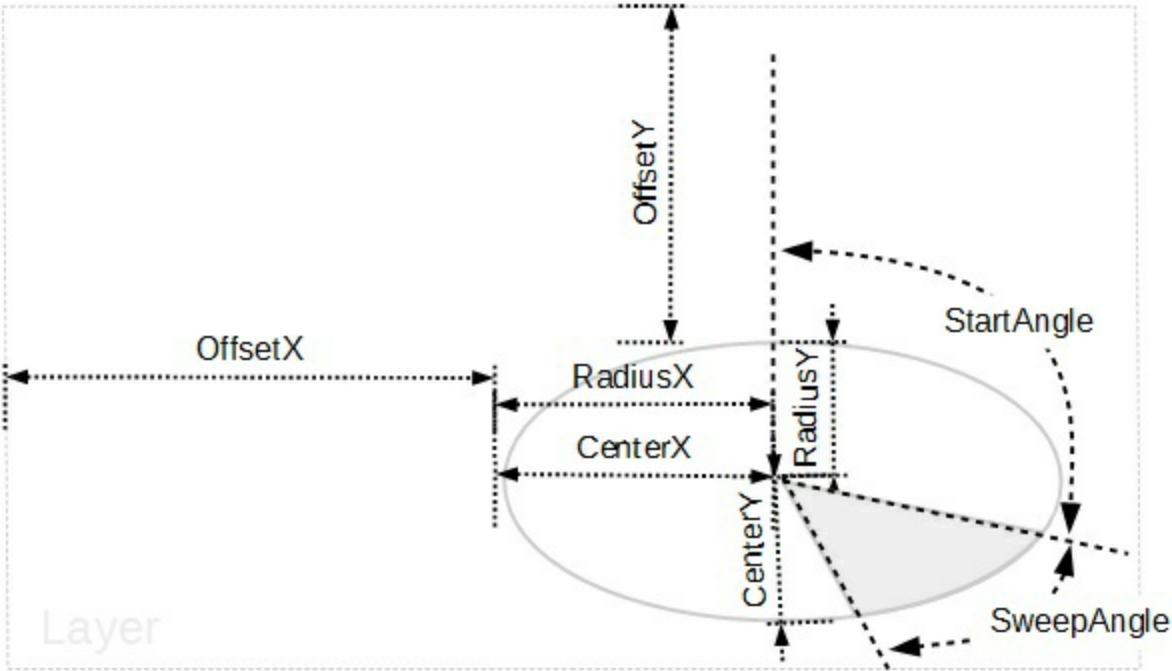
# property Clip.Pie as ClipPie

Gets access to the layer's pie clip object.

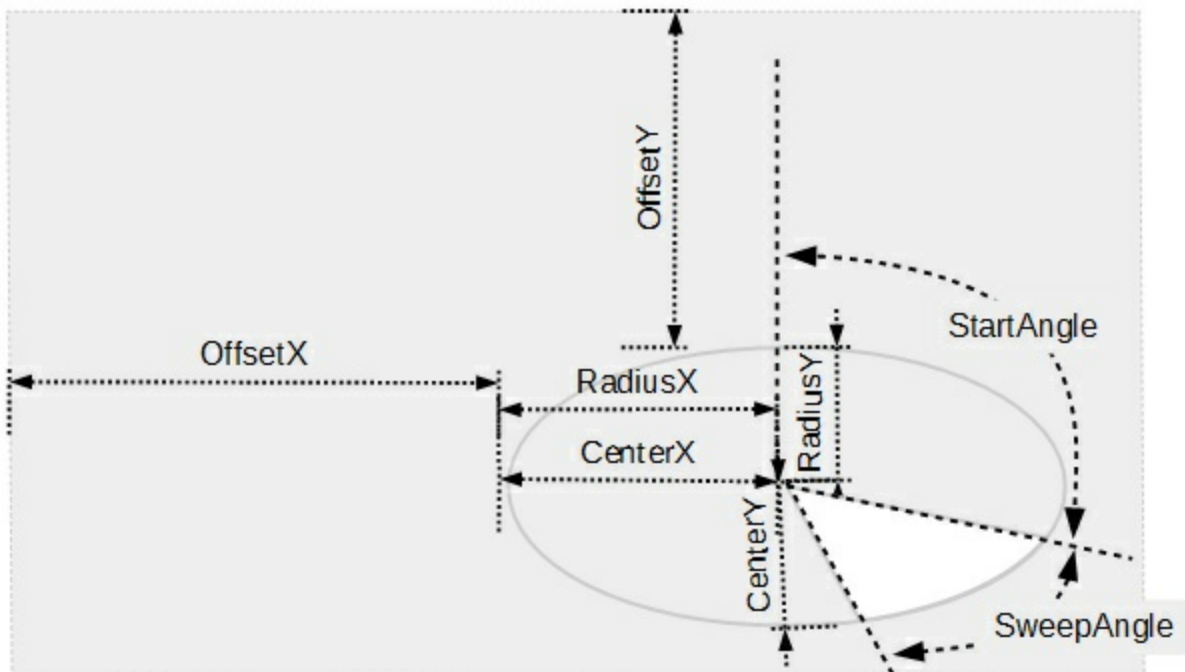
Type	Description
ClipPie	A <a href="#">ClipPie</a> object that holds information about pie clip region

The Pie property gets access to the layer's pie clip object.

The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is False, by default):



The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is True):



To define an pie clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

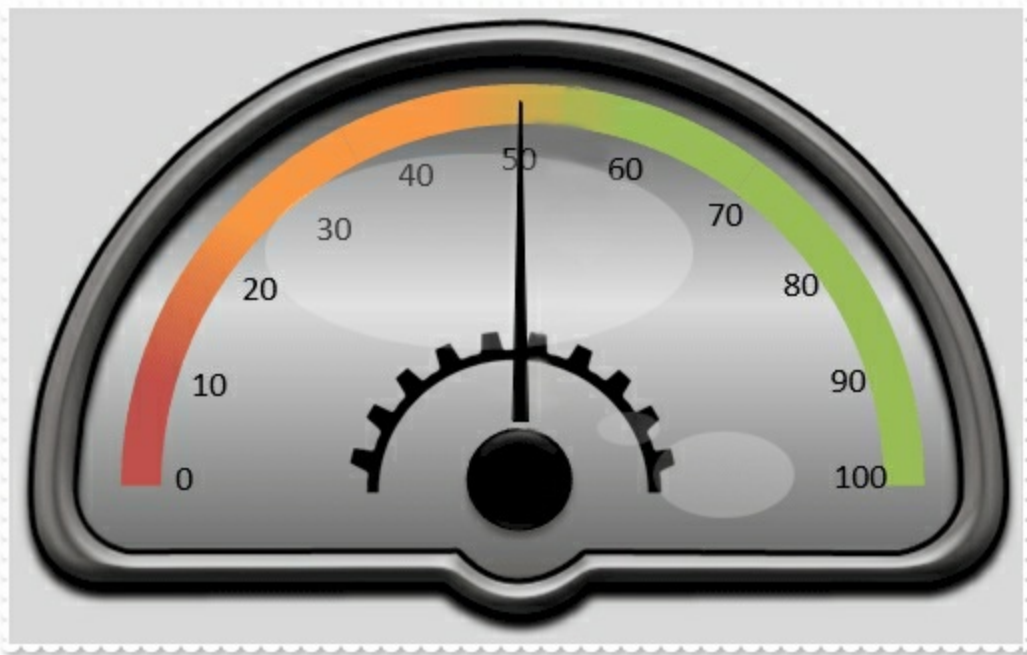
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

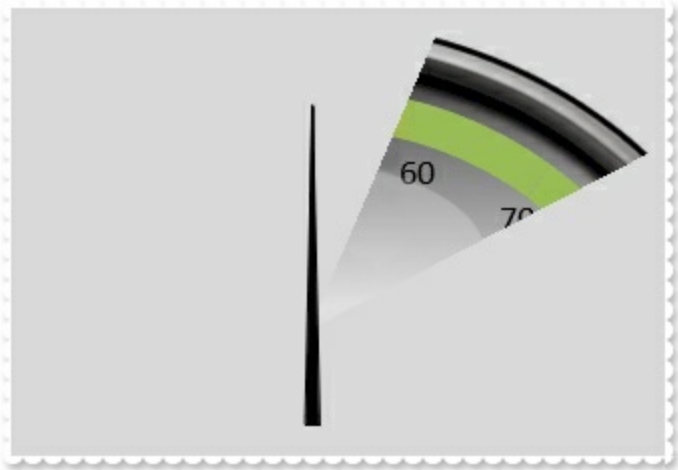
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

For instance, having the following gauge:



an pie clip region over the background layer shows as:



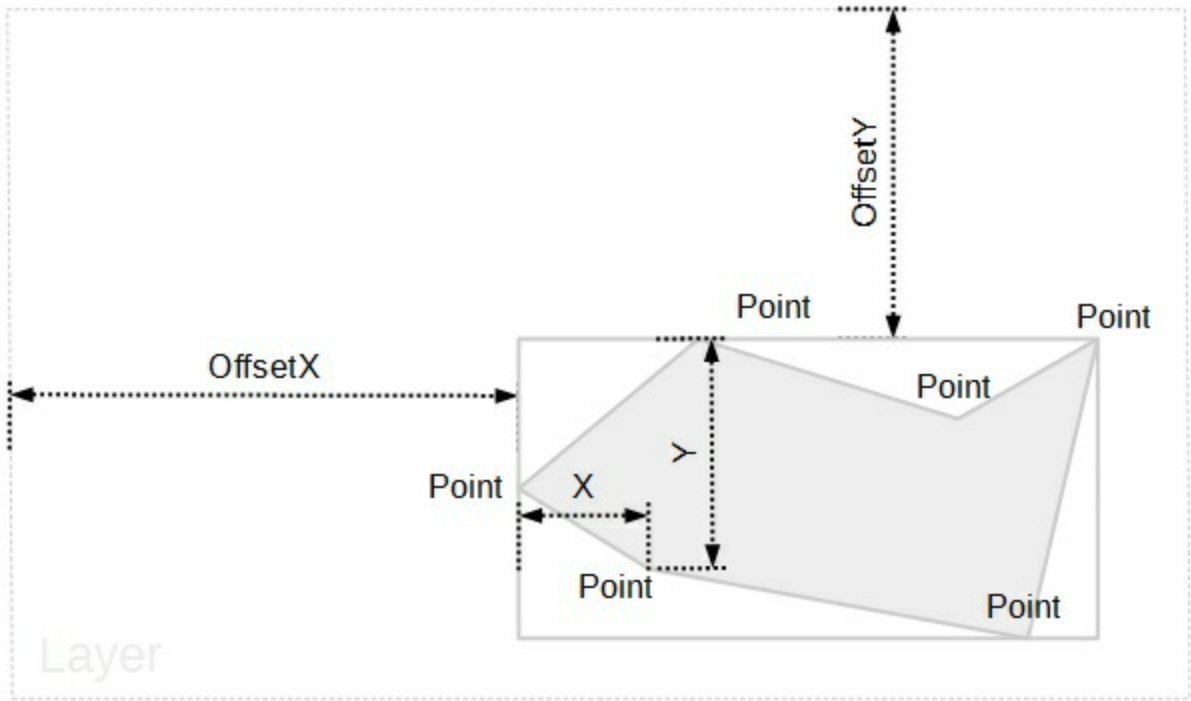
# property Clip.Polygon as ClipPolygon

Gets access to the layer's polygon clip object.

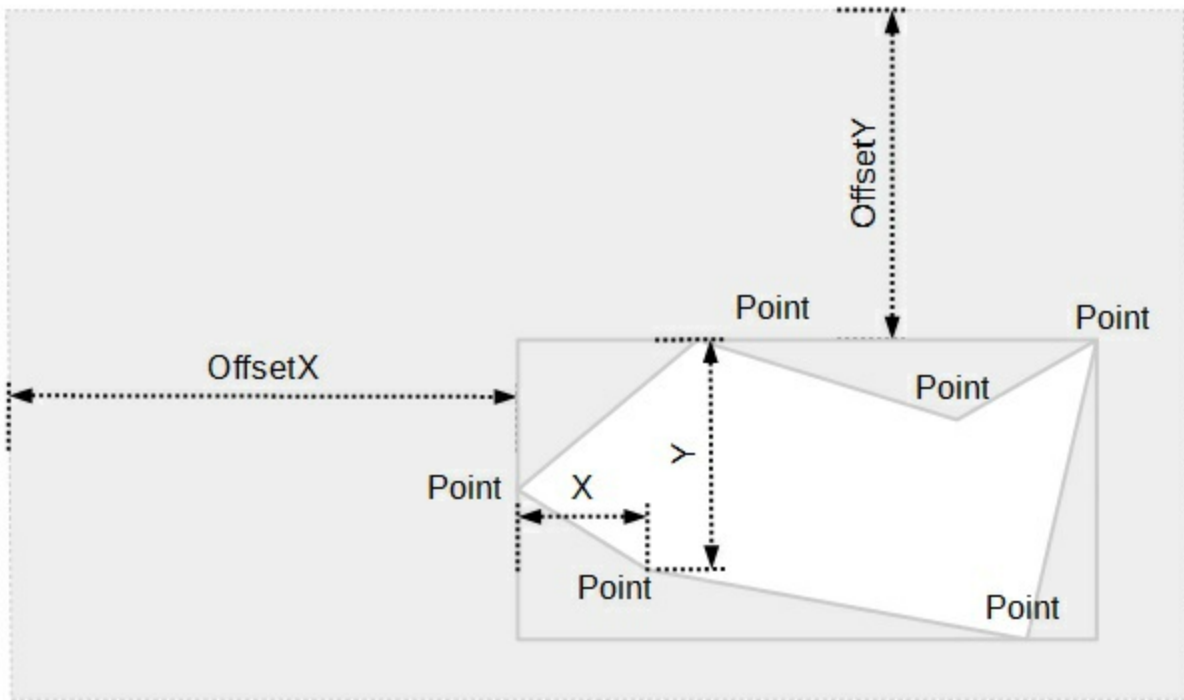
Type	Description
ClipPolygon	A <a href="#">ClipPolygon</a> object that holds information about polygonal clip region

The Polygon property gets access to the layer's polygon clip object.

The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is False, by default):



The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is True):



To define an polygonal clip region over the layer you can use any of the following properties:

- [Points](#), Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [Y](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

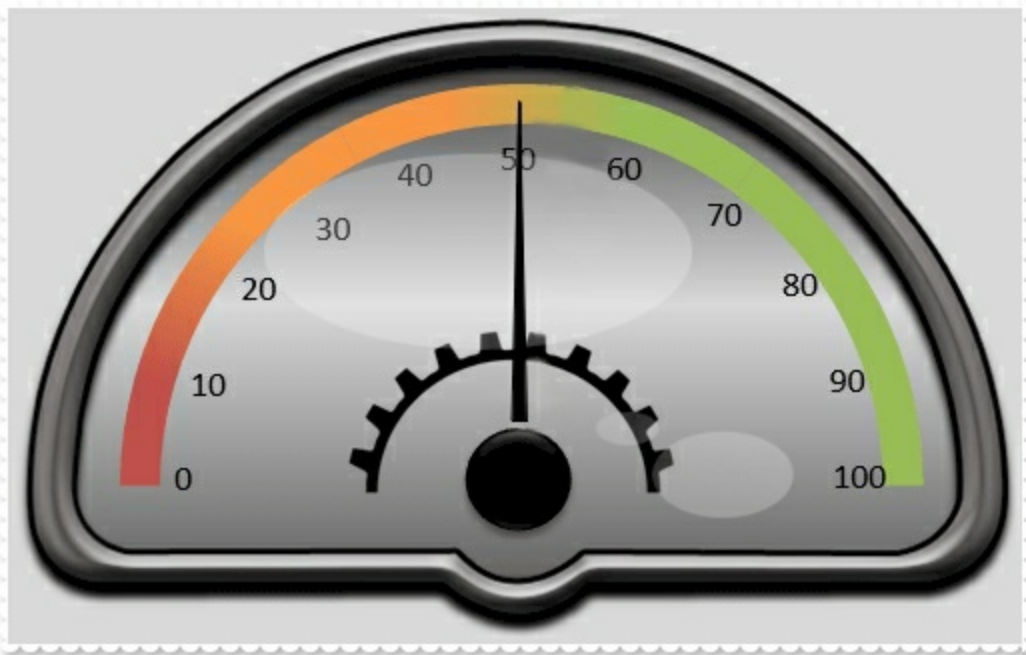
- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

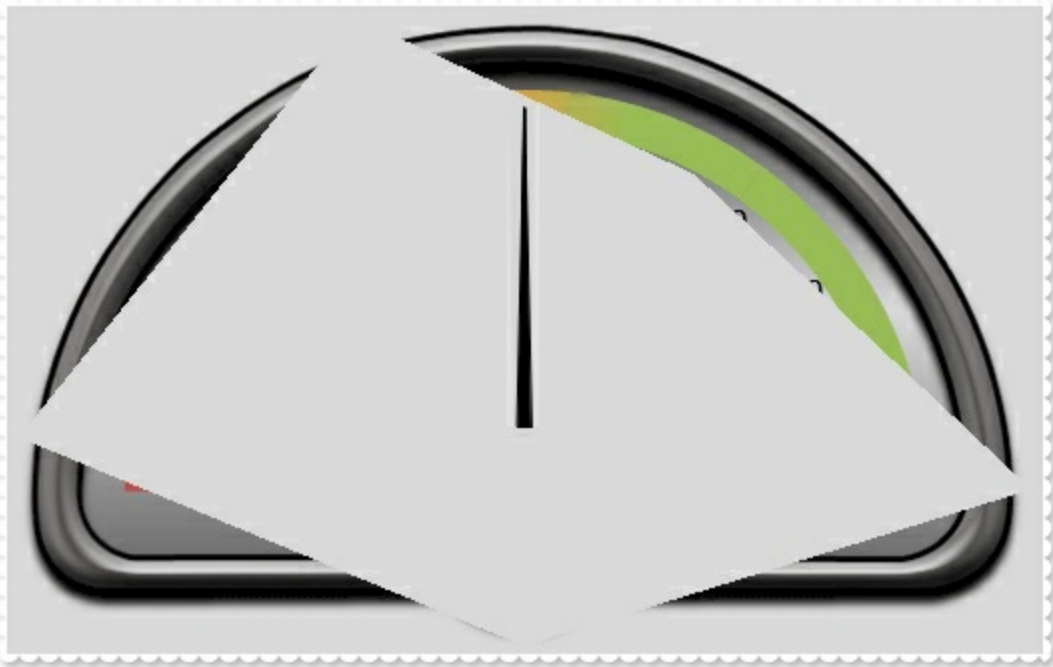
The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

For instance, having the following gauge:





an polygonal clip region over the background layer shows as:



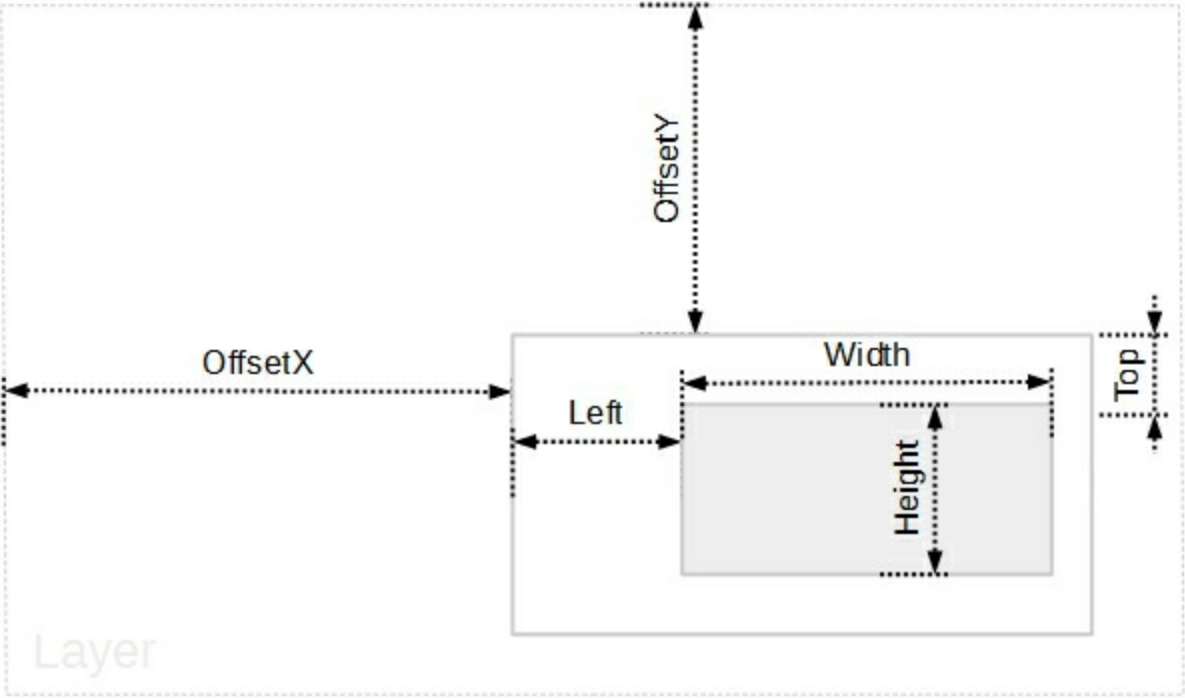
# property Clip.Rectangle as ClipRectangle

Gets access to the layer's rectangle clip object.

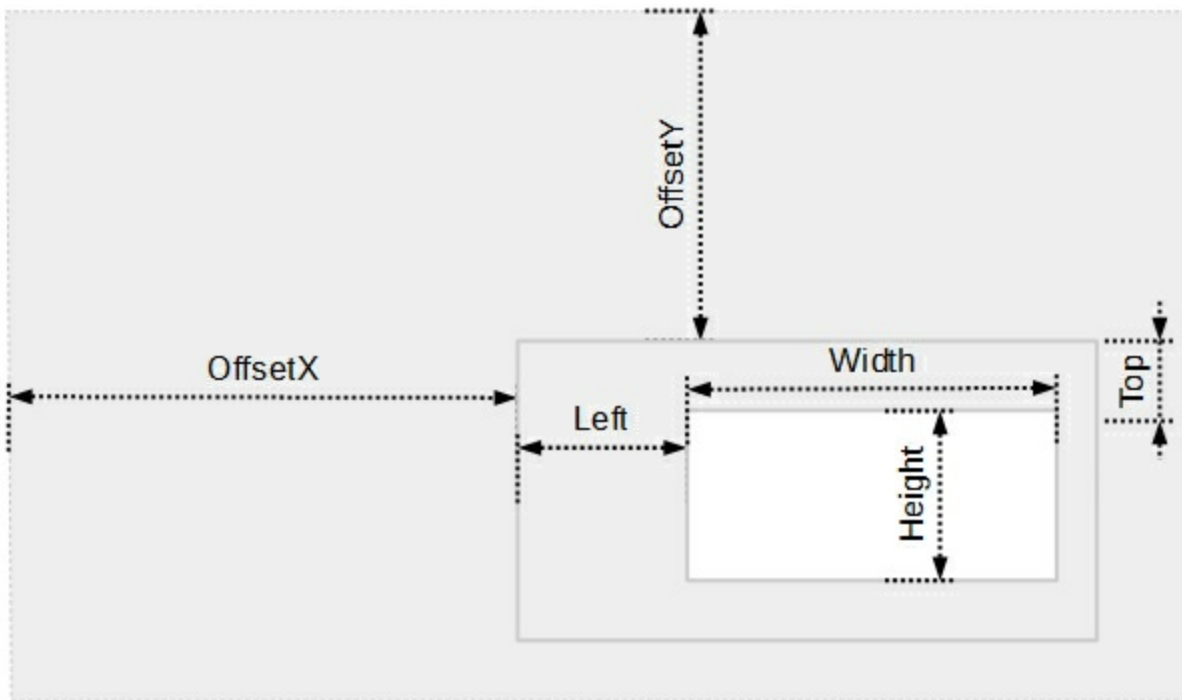
Type	Description
ClipRectangle	A <a href="#">ClipRectangle</a> object that holds information about rectangular clip region

The Rectangle property gets access to the layer's rectangular clip object.

The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is False, by default):



The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is True):



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

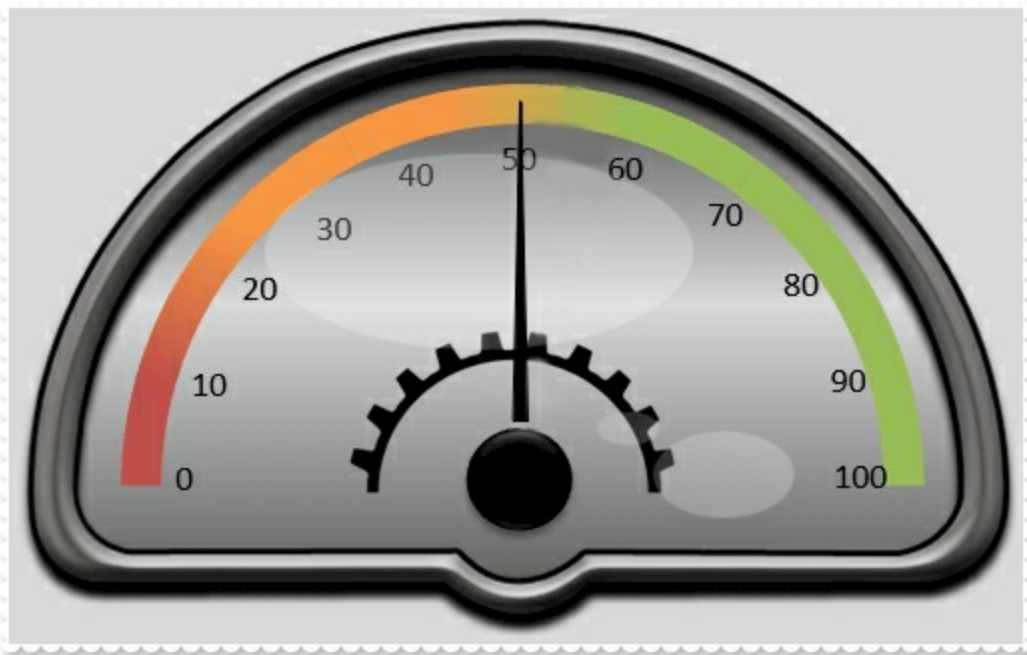
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

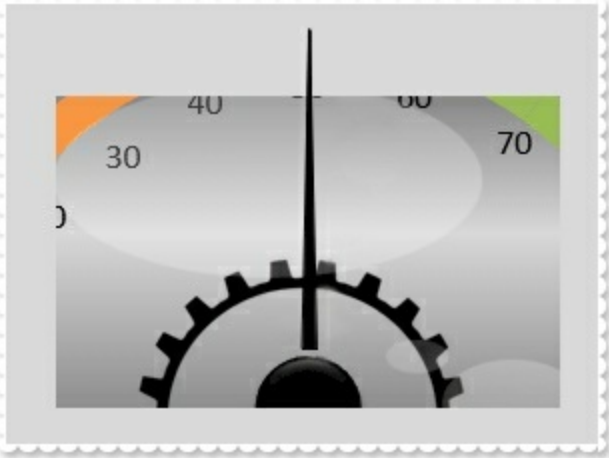
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

For instance, having the following gauge:



a rectangular clip region over the background layer shows as:



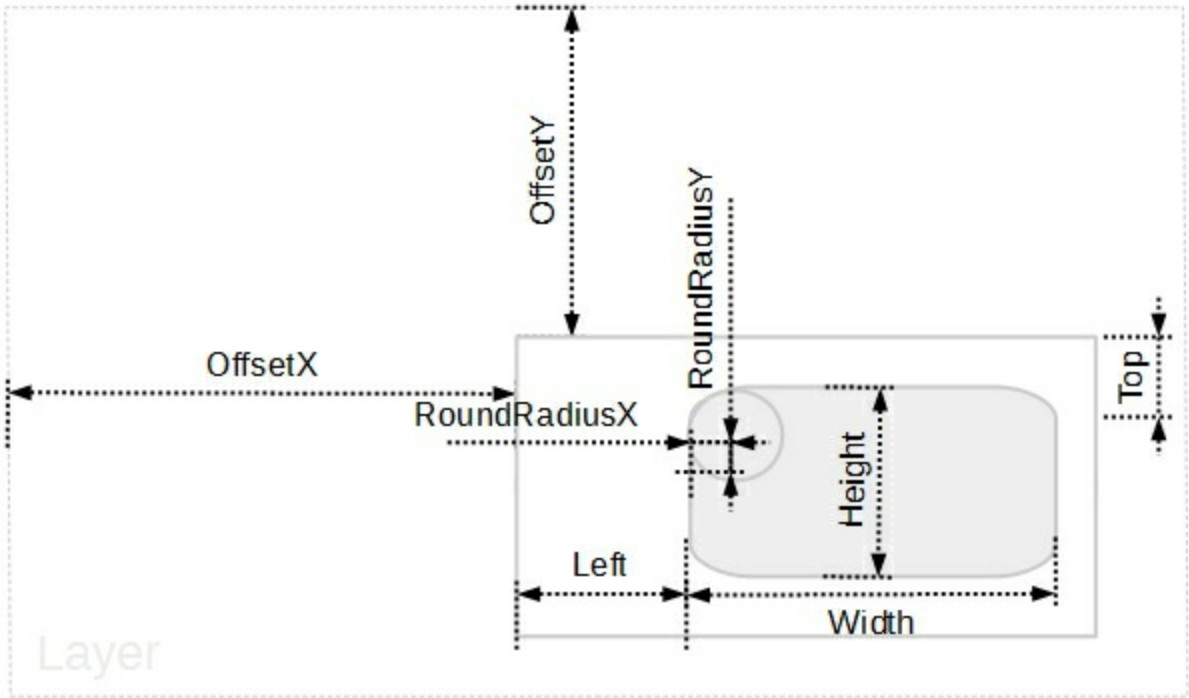
# property Clip.RoundRectangle as ClipRoundRectangle

Gets access to the layer's round rectangle clip object.

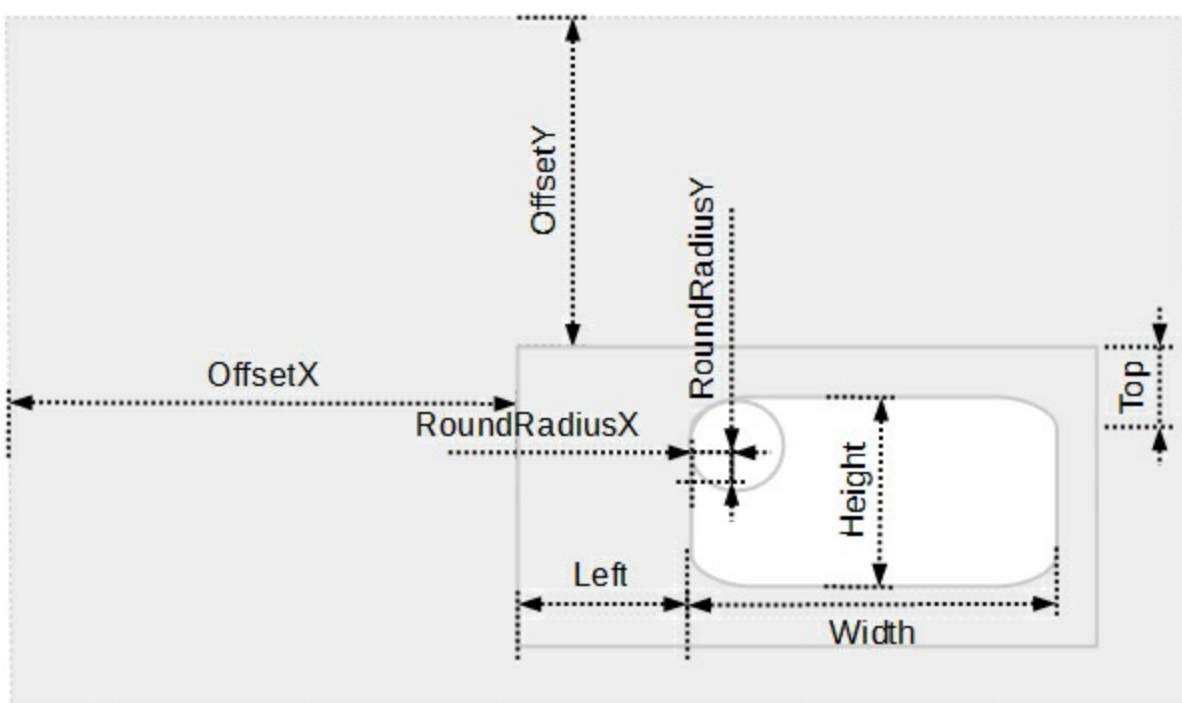
Type	Description
ClipRoundRectangle	A <a href="#">ClipRoundRectangle</a> object that holds information about rectangular clip region

The RoundRectangle property gets access to the layer's round rectangular clip object.

The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is False, by default):



The following screen shot shows properties of the clipping objects relative to the layer ([InverseClip](#) property is True):



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

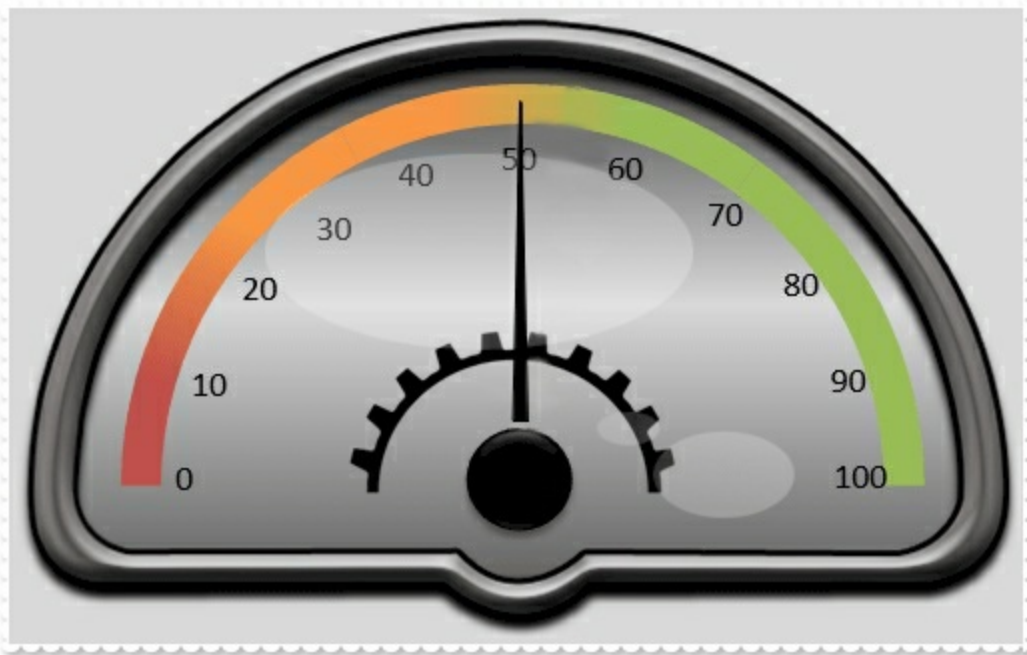
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

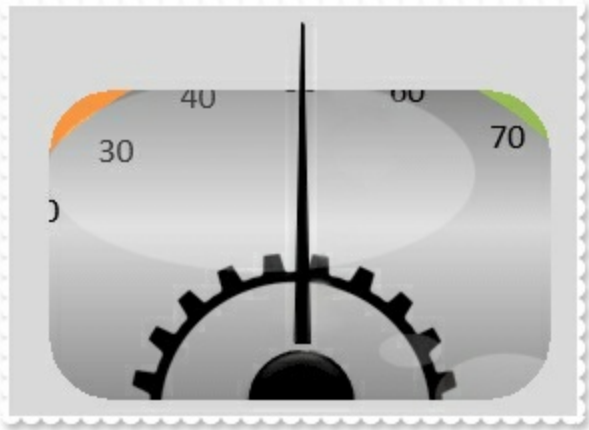
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

For instance, having the following gauge:



a round rectangular clip region over the background layer shows as:



# property Clip.Type as LayerClipTypeEnum

Specifies the type of the clipping the current layer supports.

Type	Description
<a href="#">LayerClipTypeEnum</a>	A LayerClipTypeEnum expression that specifies the combination of clipping objects that currently is applied on the layer.

By default, the Type property is exLayerClipEmpty, which indicates that no clipping is applied to the layer. The Type property can be a combination of any flag of [LayerClipTypeEnum](#) type, which indicates intersection of the clipping objects will be applied on the layer. The Type property is automatically updated as soon as any property of any clipping object is invoked. In other words, if [RadiusY](#) property of the [ClipPie](#) object is called, the Type property includes automatically the exLayerClipPie. You can use the Type property to apply / prevent clipping to be applied on the specified layer. For instance, set the Type property on exLayerClipEmpty to prevent applying any clipping on the current layer. Changing the Type property at runtime, does not remove any clipping property of any clipping object.



# property Clip.Value as Variant

Indicates the object's value.

Type	Description
Variant	A VARIANT expression to be used as a replacement of the <b>value</b> keyword in any property of Clip... objects.

By default, the Value property is empty. You can associate a value with a clipping object. For instance, let's say we define the value of a clipping rectangle as being its width. In other words, if we change the clip's Value property, the [Width](#) property of the [ClipRectangle](#) object is changed too, so the clipping rectangle will vary in its width. The same we can imagine the sweep angle or a pie or radius of a circle. The [Change](#) event notifies whether a layer is moved, rotated, so during this event we can call the clip's Value to update the clipping region on specified layer.

For instance, let's say that we have the following:



and we want to clip the "Clip" layer, as soon as the "Thumb" is rotated like follows:



so you need to do the following:

- specify the "Thumb" to be rotatable, using the [OnDrag](#) property, like `Layers.Item("Thumb").OnDrag = exDoRotate`. In case we need to specify a different initial position of the layer, we can call the [DefaultRotateAngle](#) property, to specify another angle by default.
- specify the clipping zone for "Clip" layer to be a pie, such as `Layers.Item("Clip").Clip.Pie.SweepAngle = "value"`, that specifies that the sweep angle of the clipping pie is controlled by the clip's Value property
- handle the [Change](#) event and call `Layers.Item("Clip").Clip.Value = Layers.Item("Thumb").RotateAngle`, which means that when any change occurs ( the "Thumb" is rotated ), change the Value of the Clip object of the "Clip" layer to be the rotate angle of the "Thumb" layer, so in other words as the clip's Value is associated with the sweep angle, change the sweep angle of the clipping pie to be the same as the rotation angle of the "Thumb" layer.

Now, let's say we want to remove the transparency on the "Clip" layer, so we need to add a new clipping object, this time of Picture object, with the same picture as:

- `Layers("Clip").Clip.Picture.Name = Layers("Clip").Background.Picture.Name`, which applies the clipping from the same picture, this time with no transparent pixels

so we get:



And if we hide a few intermediate layers we can get:



or if we add a new layer as a clone of others with a clipping pie we can get:



For instance, if you have:

- `Layers.Item("Clip").Clip.Value = 45`, you actually clip as a pie for a 45 degree the "Clip" layer.
- If using `Layers("Clip").Clip.Pie.SweepAngle = "2 * value"`, it indicates that the angle of the clipping region is twice the rotation angle of the thumb.
- If using `Layers("Clip").Clip.Pie.SweepAngle = "value / 2"`, it indicates that the angle of the clipping region is half of the rotation angle of the thumb.

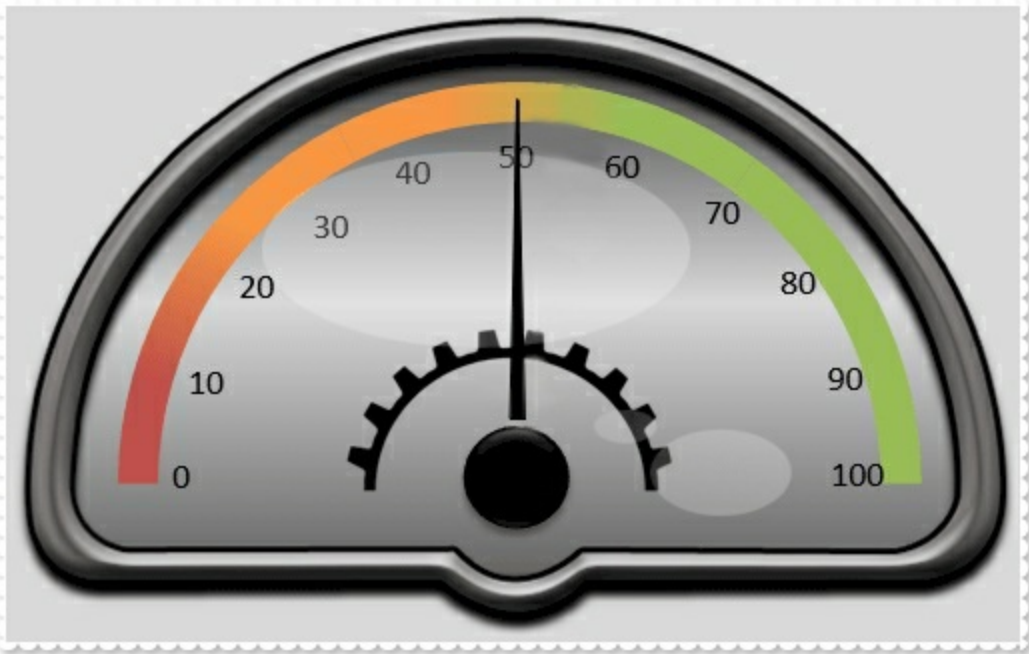
Any of the following properties ( or combination of them ) can be used to do the clipping:

- [Ellipse](#), clips the layer as a ellipse / circle
- [Picture](#), clips the layer using a picture as a mask
- [Pie](#), clips the layer as a arc / pie
- [Polygon](#), clips the layer giving the points that define a polygon, triangle, rectangle, and so on
- [Rectangle](#), clips the layer giving a rectangle
- [RoundRectangle](#), clips the layer giving a round rectangle

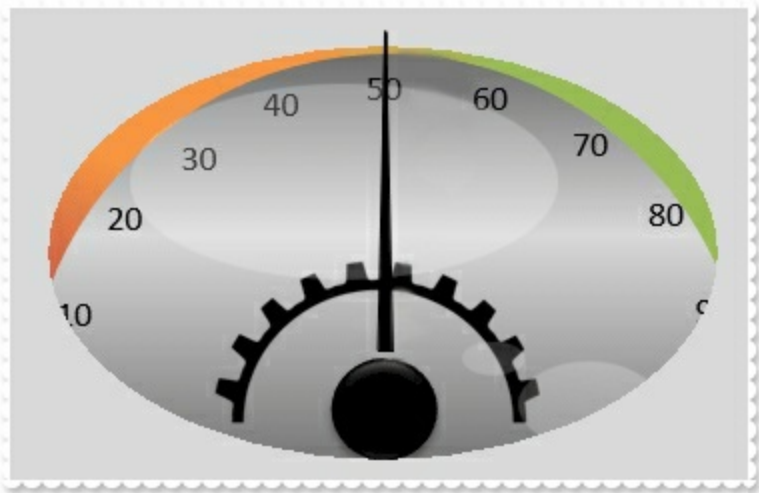
# ClipEllipse object

The ClipEllipse object holds information about an elliptical clip region.

For instance, having the following gauge:



an elliptical clip region over the background layer shows as:



The ClipEllipse supports the following properties and methods:

Name	Description
<a href="#">CenterX</a>	Specifies the x-position / expression of the center of the clip, relative to the layer.
<a href="#">CenterY</a>	Specifies the y-position / expression of the center of the clip, relative to the layer.
<a href="#">InverseClip</a>	Indicates whether the current clip object is inverted.

[OffsetX](#)

Specifies the x-offset expression / value of the clip, relative to the layer.

[OffsetY](#)

Specifies the y-offset expression / value of the clip, relative to the layer.

[RadiusX](#)

Specifies the x-radius value / expression of the clip, relative to the layer.

[RadiusY](#)

Specifies the y-radius value / expression of the clip, relative to the layer.

---

# property ClipEllipse.CenterX as String

Specifies the x-position / expression of the center of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-position / expression of the center of the clip, relative to the layer.

By default, the CenterX property is empty, which indicates the center of the layer The CenterX property is "width/2" ( center of the layer ), if the expression is missing or invalid.

For instance:

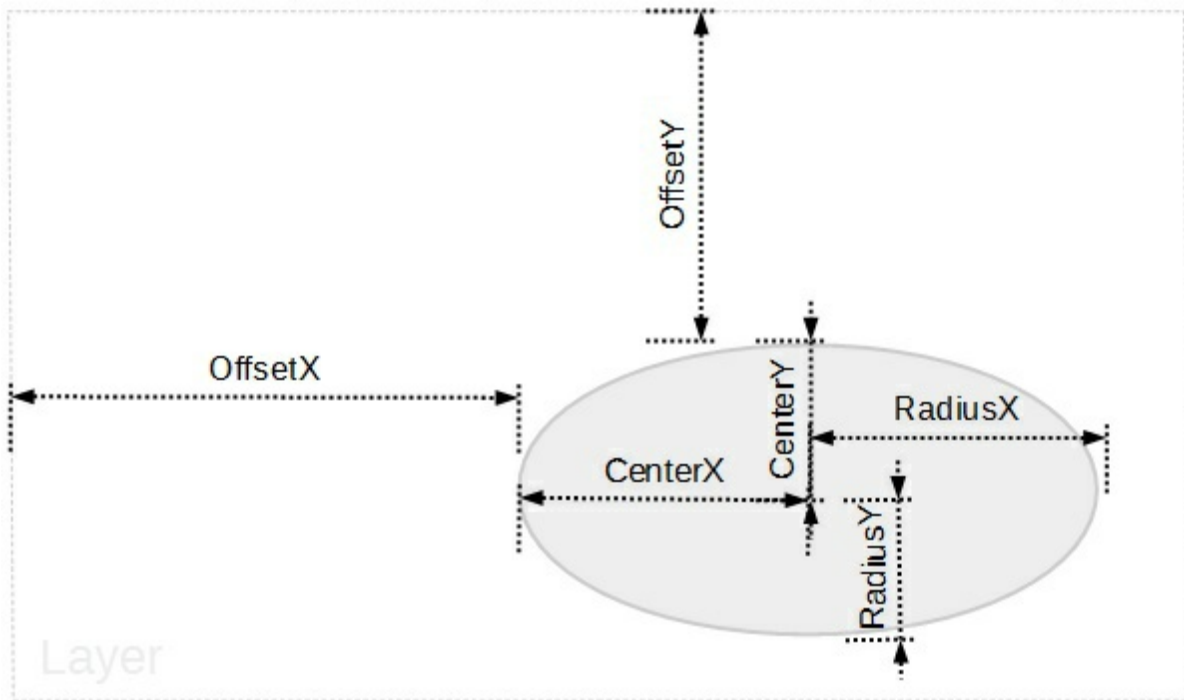
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- **CenterX**, Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipEllipse.CenterY as String

Specifies the y-position / expression of the center of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-position / expression of the center of the clip, relative to the layer.

By default, the CenterY property is empty, which indicates the center of the layer The CenterY property is "height/2" ( center of the layer ), if the expression is missing or invalid.

For instance:

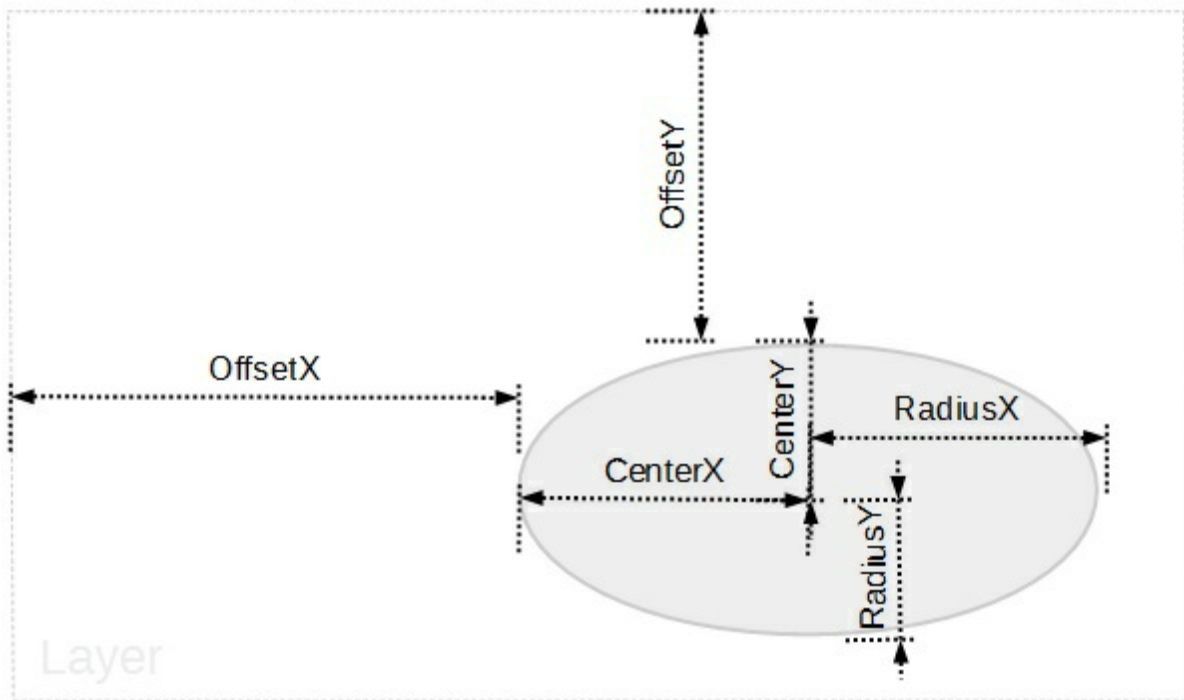
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or right side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipEllipse.InverseClip as Boolean

Indicates whether the current clip object is inverted.

Type	Description
Boolean	A Boolean expression that indicates whether the current clip object is inverted.

By default, InverseClip property is False. The InverseClip property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

To define an elliptical clip region over the layer you can use any of the following properties:

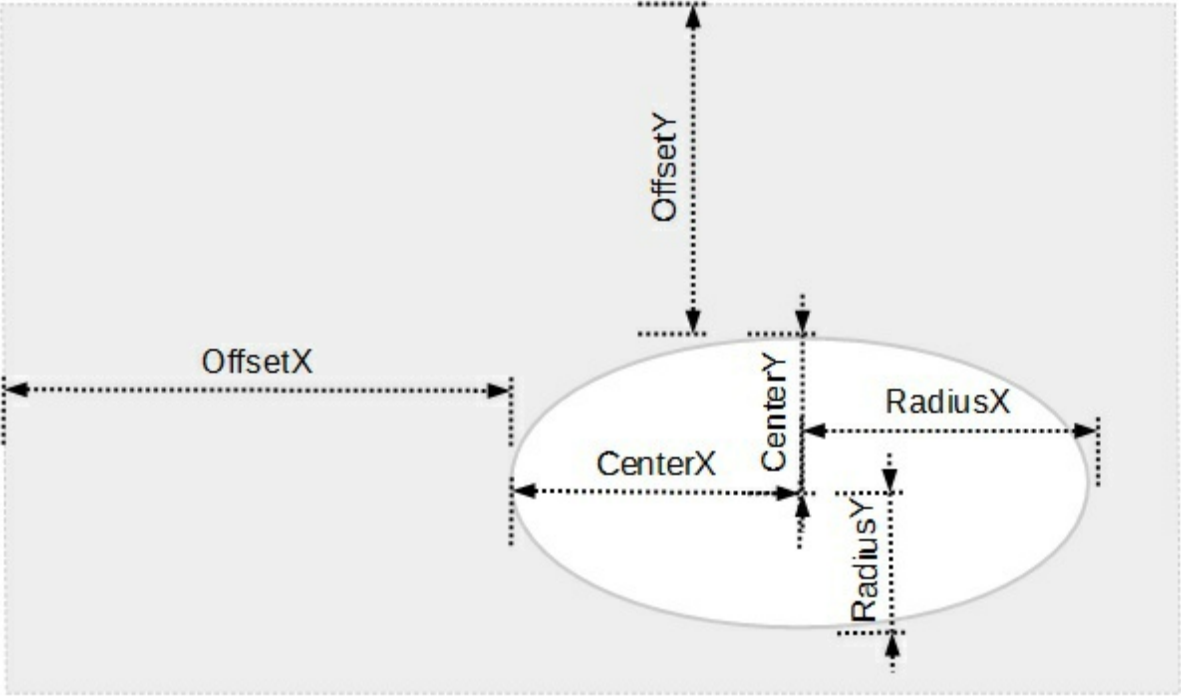
- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:



# property ClipEllipse.OffsetX as String

Specifies the x-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-offset expression / value of the clip, relative to the layer.

By default, the OffsetX property is empty, which indicates the value of 0 ( left side of the layer ). The OffsetX property is 0, if the expression is missing or invalid.

For instance:

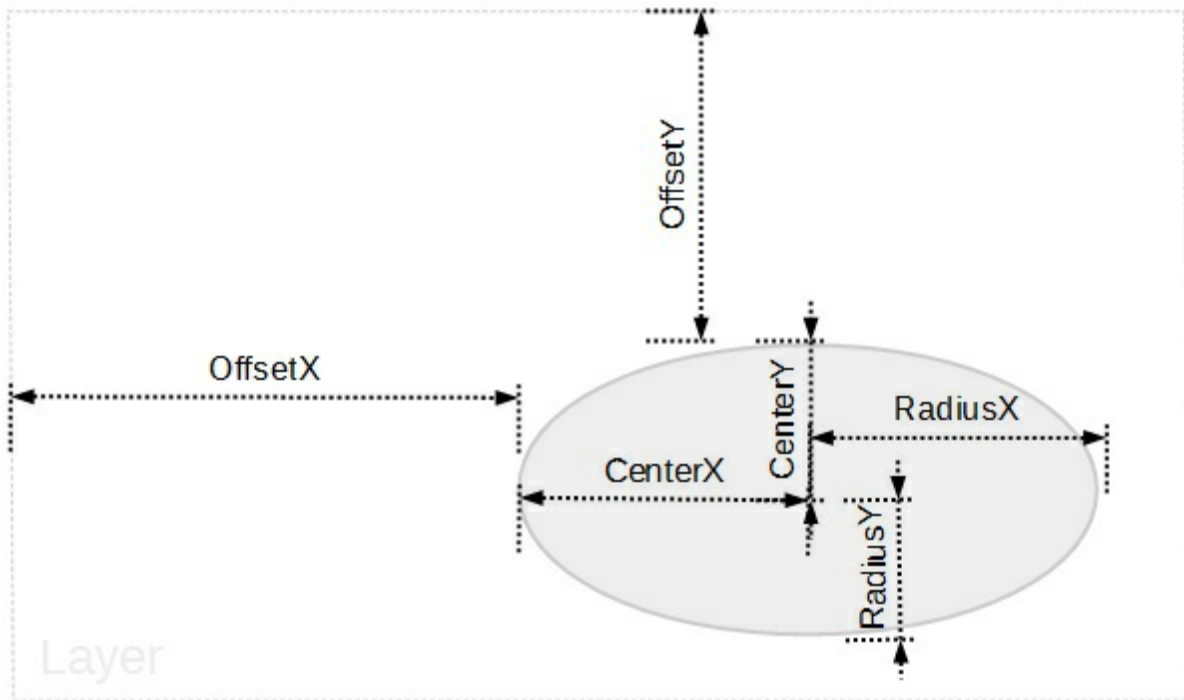
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipEllipse.OffsetY as String

Specifies the y-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-offset expression / value of the clip, relative to the layer.

By default, the OffsetY property is empty, which indicates the value of 0 ( top side of the layer ). The OffsetY property is 0, if the expression is missing or invalid.

For instance:

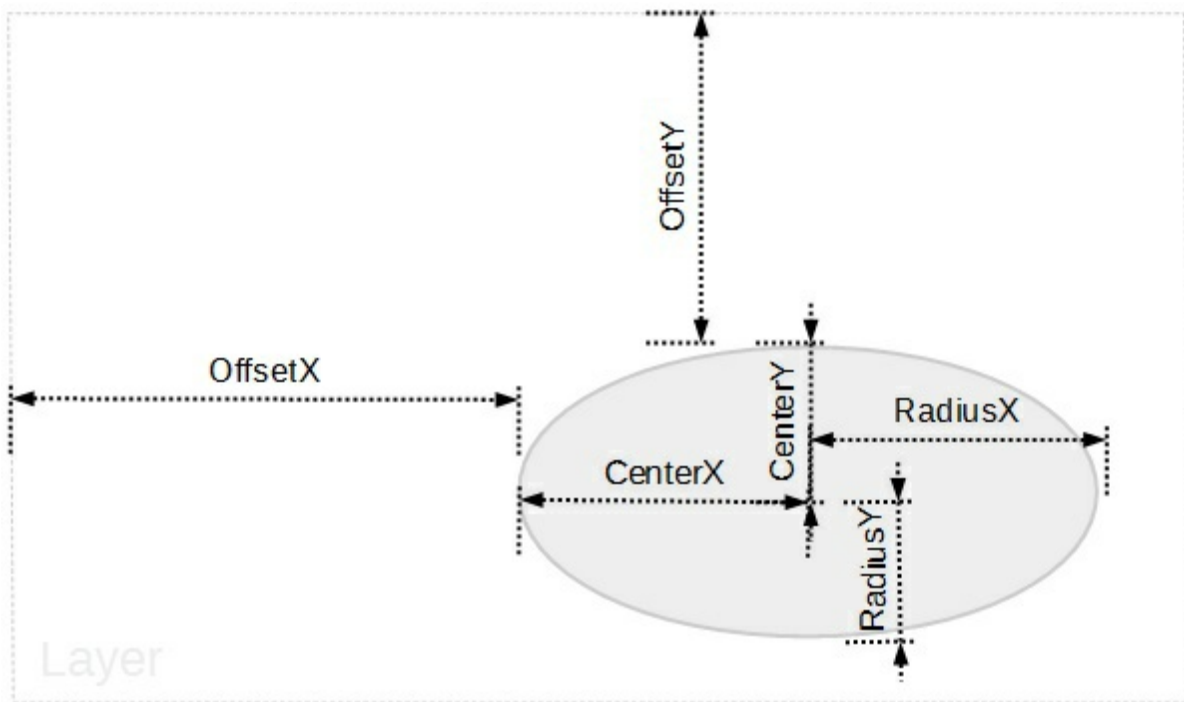
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipEllipse.RadiusX as String

Specifies the x-radius value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-radius value / expression of the clip, relative to the layer.

By default, the RadiusX property is empty, which indicates the half of the layer's width. The RadiusX property is "width/2" ( half of the layer's width ), if the expression is missing or invalid.

For instance:

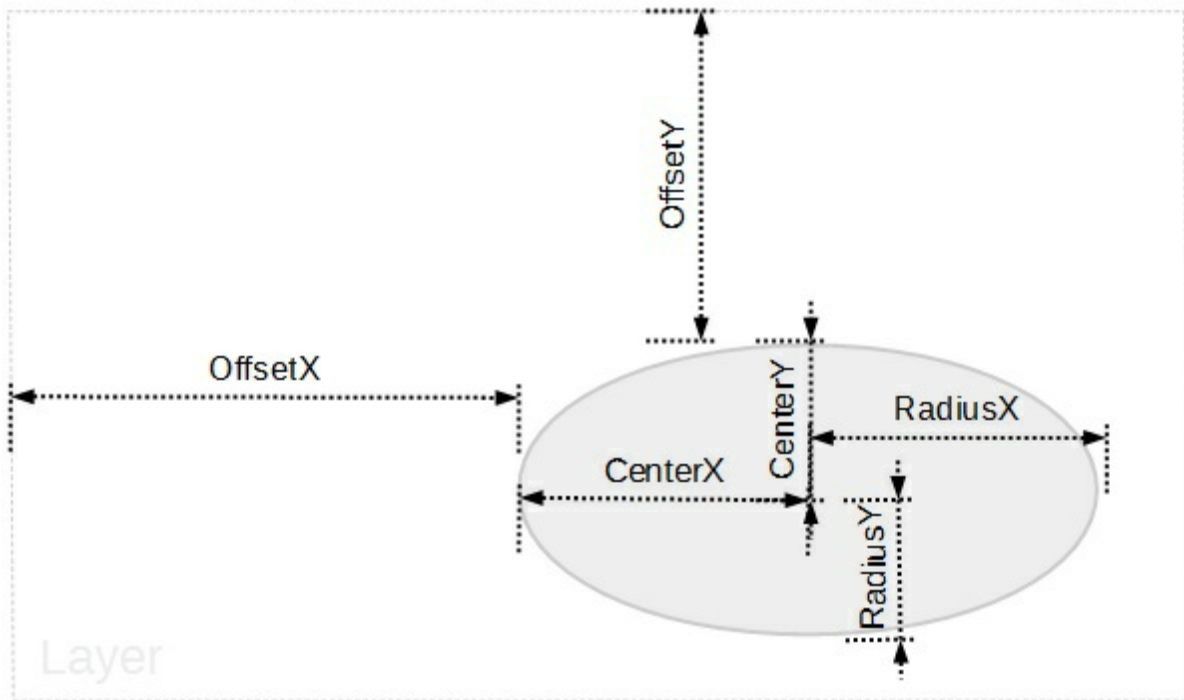
- "15", 15 pixels radius
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipEllipse.RadiusY as String

Specifies the y-radius value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-radius value / expression of the clip, relative to the layer.

By default, the RadiusY property is empty, which indicates the half of the layer's height. The RadiusY property is "height/2" ( half of the layer's height ), if the expression is missing or invalid.

For instance:

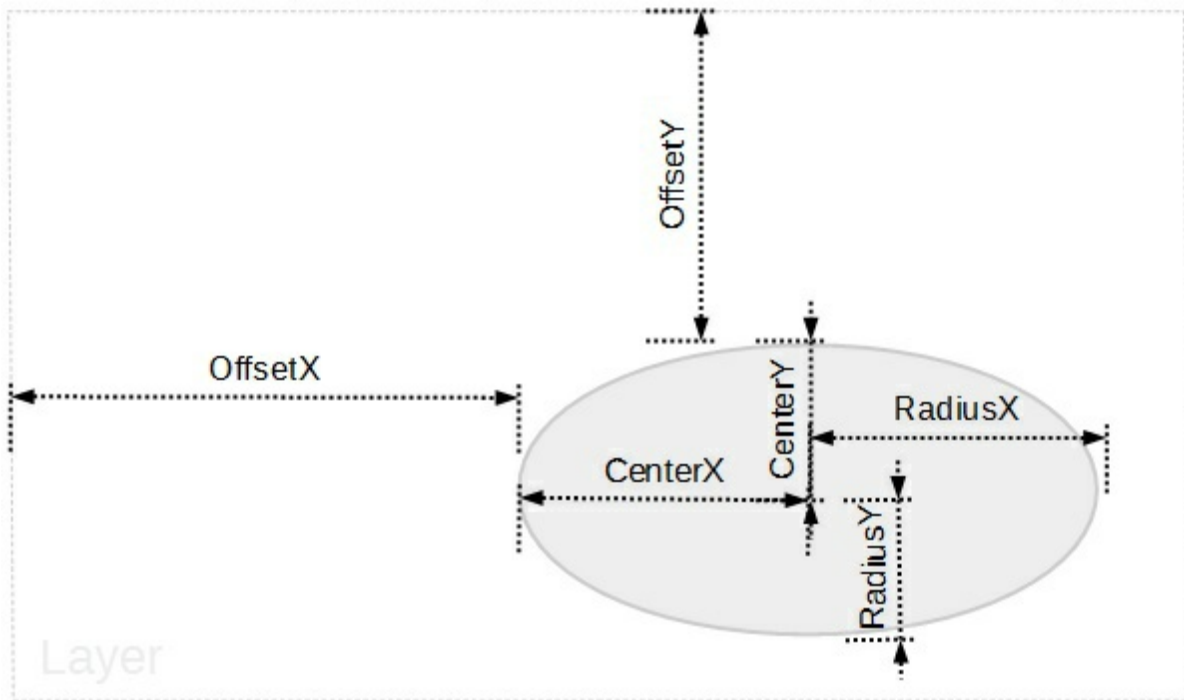
- "15", 15 pixels radius
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or right side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

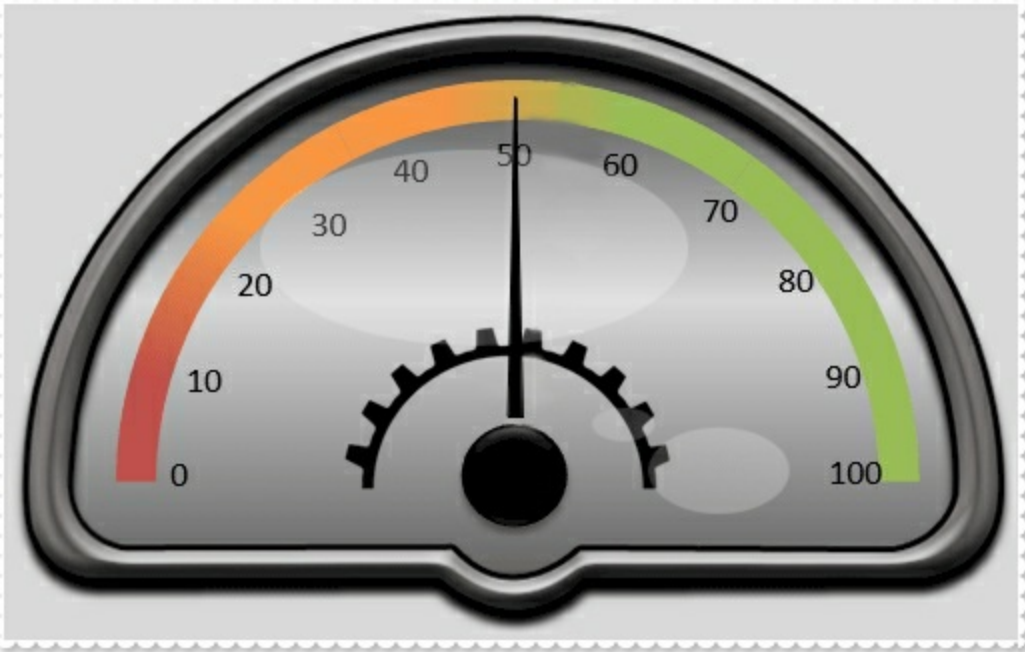
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# ClipPicture object

The ClipPicture object holds information about an picture clip region.

For instance, having the following gauge:



a picture clip region over the background layer shows as:



when the source picture is:



The ClipPicture property supports the following properties and methods:

Name	Description
<a href="#">AlphaFrom</a>	Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
<a href="#">AlphaTo</a>	Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

[DisplayAs](#)

Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.

[Height](#)

Specifies the height value / expression of the clip, relative to the layer.

[InverseClip](#)

Indicates whether the current clip object is inverted.

[Left](#)

Specifies the left position / expression of the clip, relative to the layer.

[Name](#)

Indicates the picture to be used as a mask/clip.

[OffsetX](#)

Specifies the x-offset expression / value of the clip, relative to the layer.

[OffsetY](#)

Specifies the y-offset expression / value of the clip, relative to the layer.

[Top](#)

Specifies the top position / expression of the clip, relative to the layer.

[Width](#)

Specifies the width value / expression of the clip, relative to the layer.

# property ClipPicture.AlphaFrom as String

Gets or sets a value that specifies the alpha-byte to start clipping the picture from.

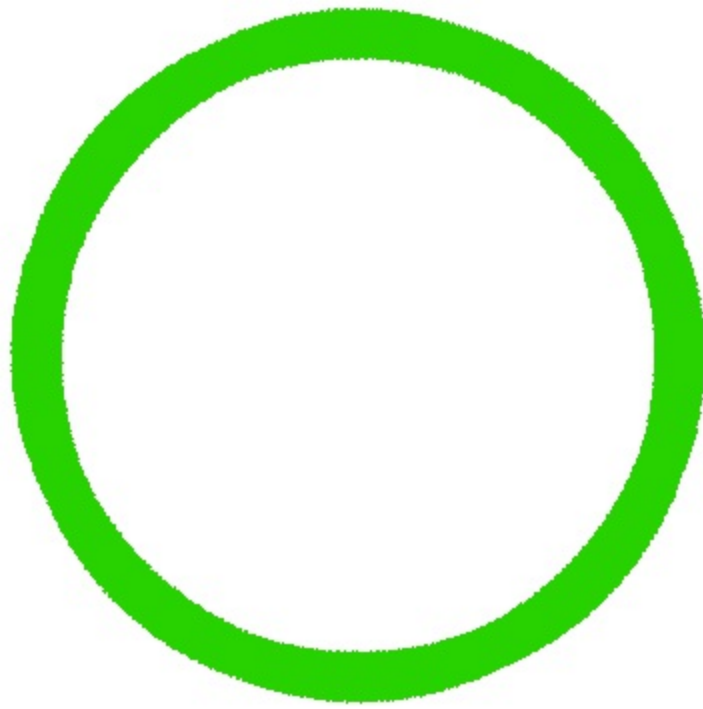
Type	Description
String	A String expression that defines the alpha-byte to start clipping the picture from.

By default, the AlphaFrom and [AlphaTo](#) properties are empty. While AlphaFrom property is empty, missing or invalid, 0 is used instead. While AlphaTo property is empty, missing or invalid, 254 is used instead. In other words, any pixel in the picture with the transparency-byte on 255 defines the clipping region ( by default, the opaque-pixels in the picture defines the clipping region. ). Use the AlphaFrom / AlphaTo to include semi-transparent pixels in the clipping region. The picture being used as a clipping region must support transparency / alpha blending ( picture's attribute includes the PICTURE\_TRANSPARENT ). For instance, you can use any PNG file with transparency. The [DisplayAs](#) property retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.

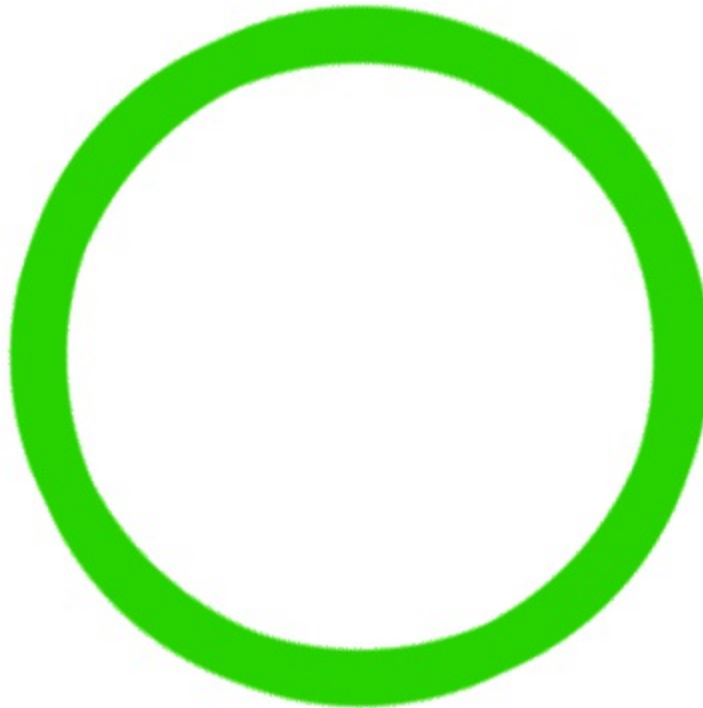
For instance, having a picture like follows:



if we apply this picture as a clipping, we get something like ( includes pixels with alpha-blend byte on 255 only, opaque-pixels ):

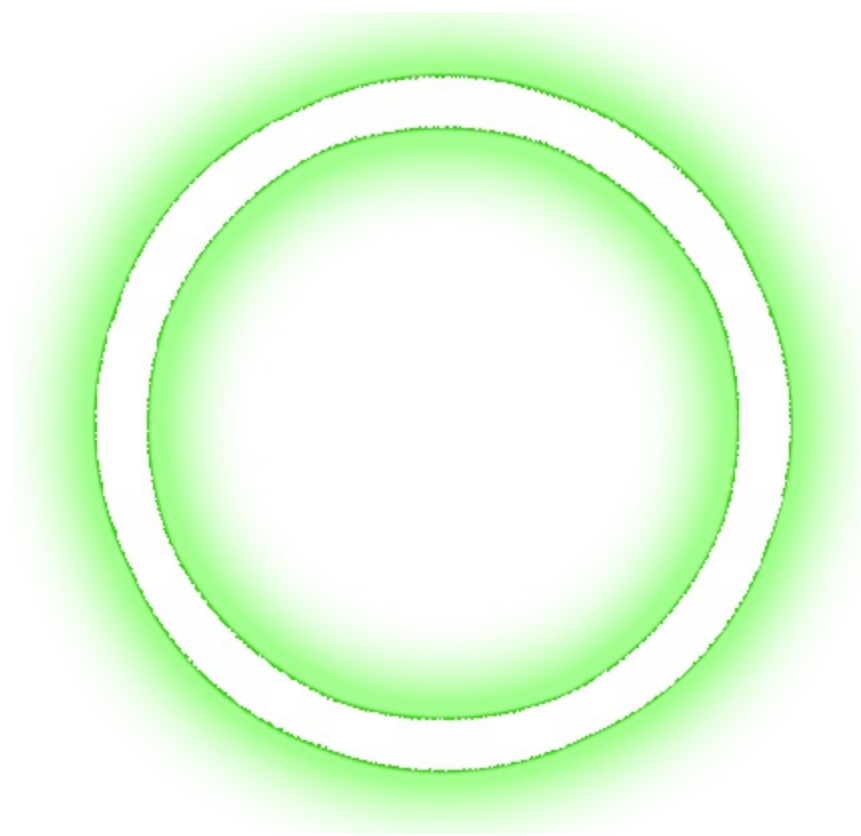


while if we change the AlphaTo field to 128 we can get something like ( includes pixels with alpha-blend byte between 129 and 255, opaque plus semi-transparent pixels to 128 ):

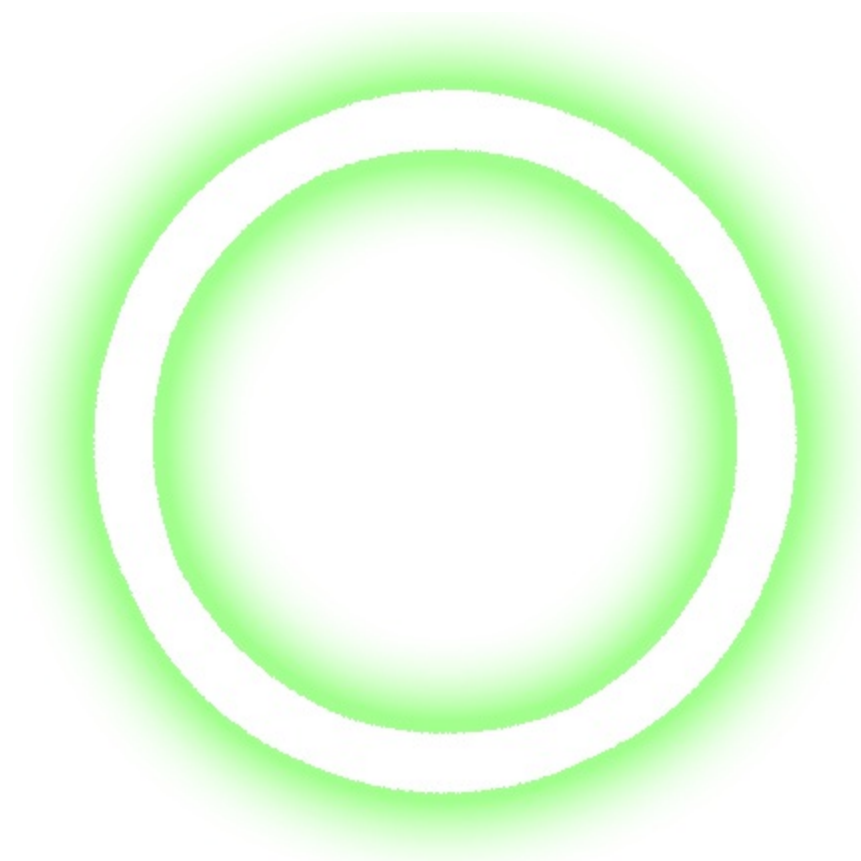


or if we change the AlphaFrom and Alpha fields to 255, we can get something like ( includes pixels with alpha-blend byte between 0 and 254, excludes the opaque pixels ):





or if we change the AlphaFrom to 128, and Alpha field to 255, we can get something like ( includes pixels with alpha-blend byte between 0 and 127, excludes the opaque pixels, and semi-transparent pixels to 128 ):



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.

- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

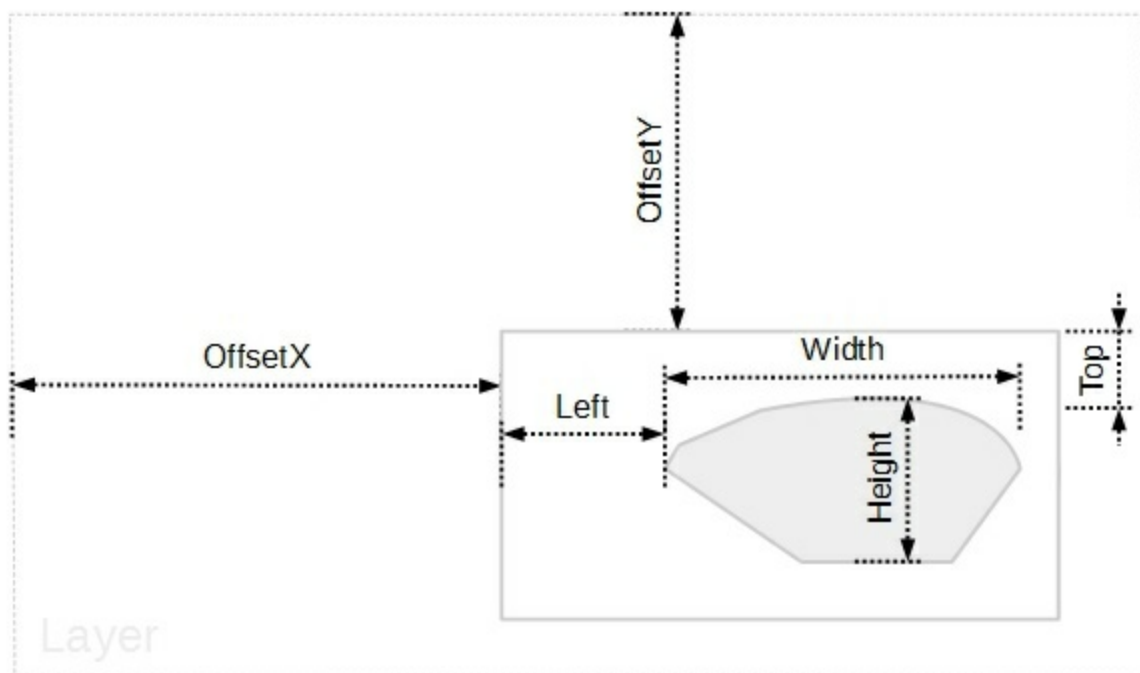
- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:



The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

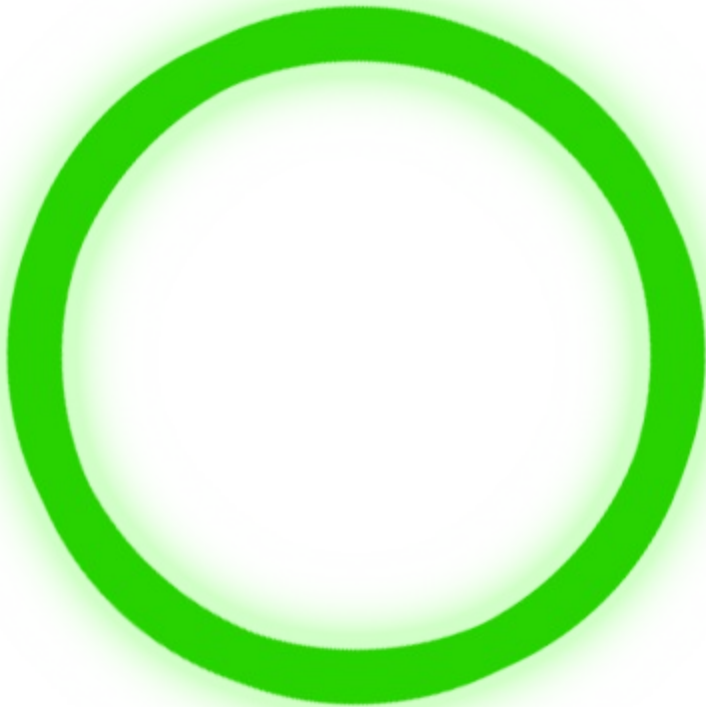
# property ClipPicture.AlphaTo as String

Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

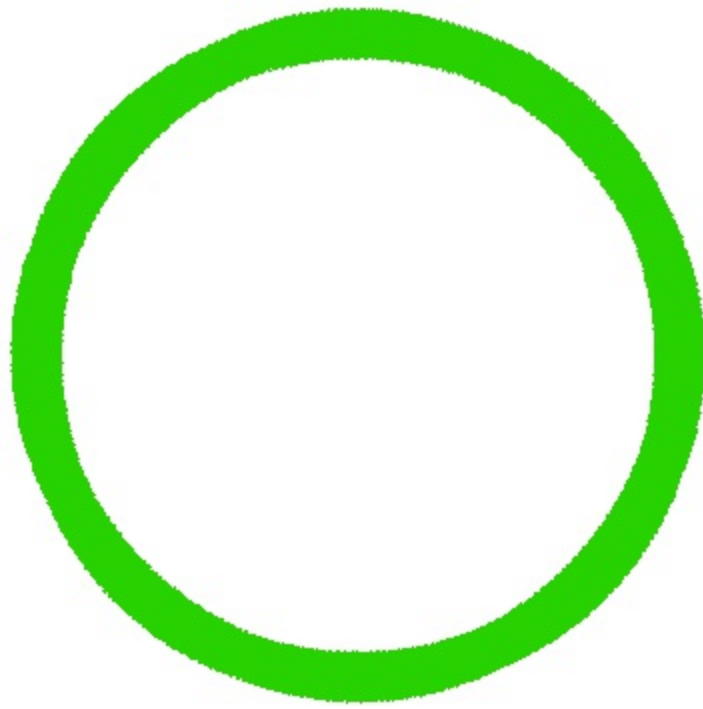
Type	Description
String	A String expression that specifies the alpha-byte to end clipping the picture to.

By default, the [AlphaFrom](#) and AlphaTo properties are empty. While AlphaFrom property is empty, missing or invalid, 0 is used instead. While AlphaTo property is empty, missing or invalid, 254 is used instead. In other words, any pixel in the picture with the transparency-byte on 255 defines the clipping region ( by default, the opaque-pixels in the picture defines the clipping region. ). Use the AlphaFrom / AlphaTo to include semi-transparent pixels in the clipping region. The picture being used as a clipping region must support transparency / alpha blending ( picture's attribute includes the PICTURE\_TRANSPARENT ). For instance, you can use any PNG file with transparency. The [DisplayAs](#) property retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.

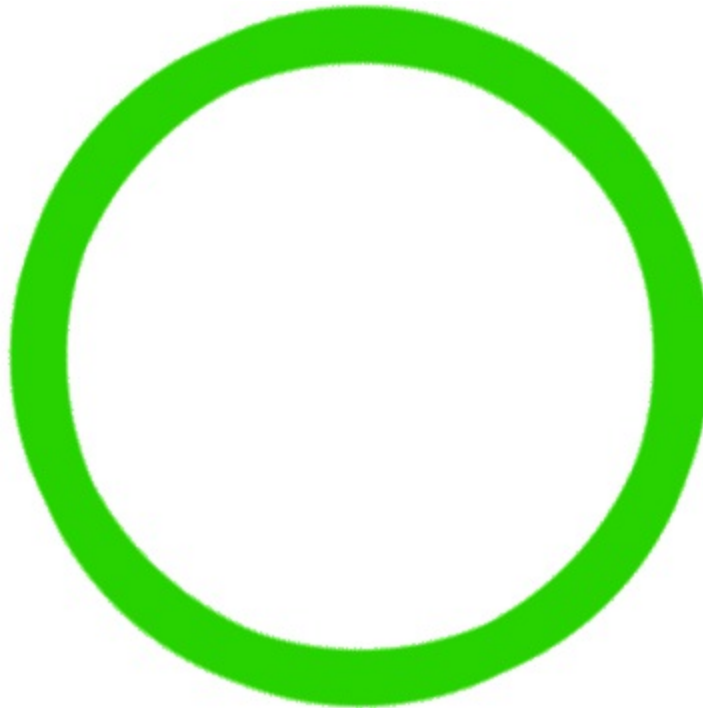
For instance, having a picture like follows:



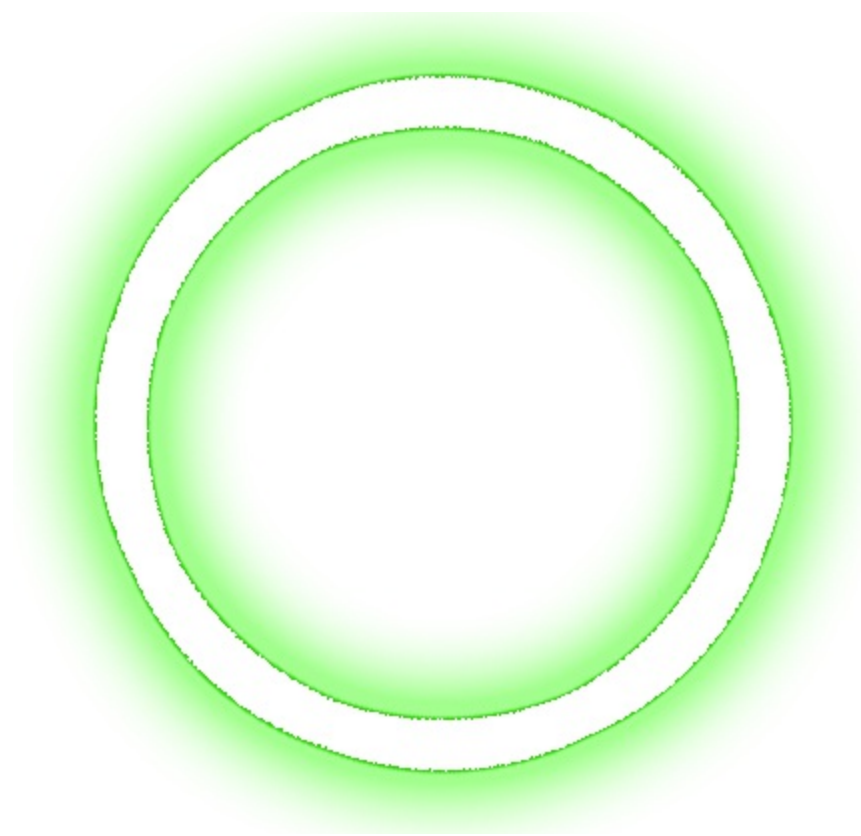
if we apply this picture as a clipping, we get something like ( includes pixels with alpha-blend byte on 255 only, opaque-pixels ):



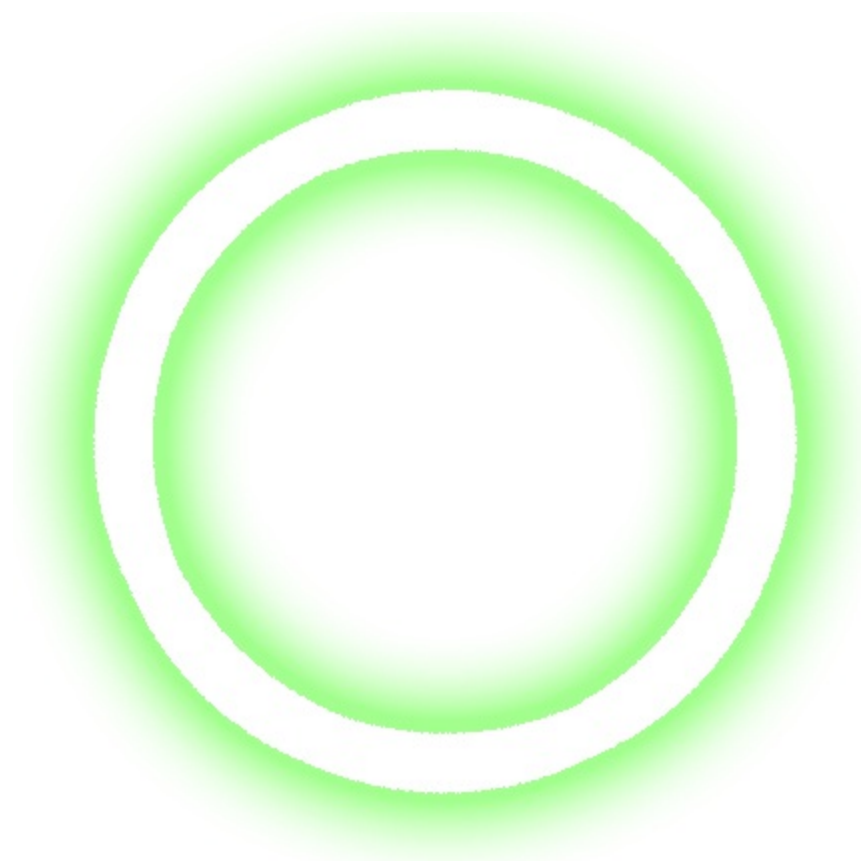
while if we change the AlphaTo field to 128 we can get something like ( includes pixels with alpha-blend byte between 129 and 255, opaque plus semi-transparent pixels to 128 ):



or if we change the AlphaFrom and Alpha fields to 255, we can get something like ( includes pixels with alpha-blend byte between 0 and 254, excludes the opaque pixels ):



or if we change the AlphaFrom to 128, and Alpha field to 255, we can get something like ( includes pixels with alpha-blend byte between 0 and 127, excludes the opaque pixels, and semi-transparent pixels to 128 ):



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.

- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

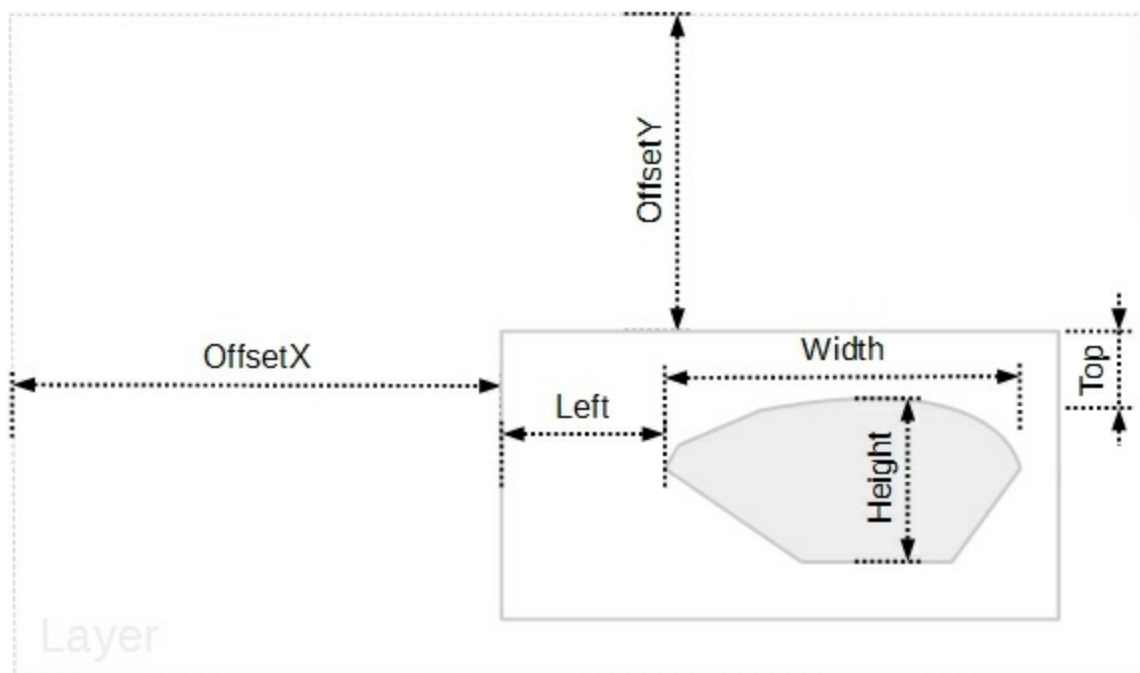
- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:



The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPicture.DisplayAs as PictureDisplayEnum

Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.

Type	Description
<a href="#">PictureDisplayEnum</a>	A PictureDisplayEnum expression that indicates the way how the graphic is arranged on the mask/clip.

By default, the DisplayAs property is Stretch. The DisplayAs property retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip. The [Name](#) property loads a picture into the clip. The picture being used as a clipping region must support transparency / alpha blending ( picture's attribute includes the PICTURE\_TRANSPARENT ). For instance, you can use any PNG file with transparency. By default, the opaque-pixels in the picture defines the clipping region. Use the [AlphaFrom](#) / [AlphaTo](#) to include semi-transparent pixels in the clipping region.

To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- DisplayAs, Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- Height, Specifies the height value / expression of the clip, relative to the layer.

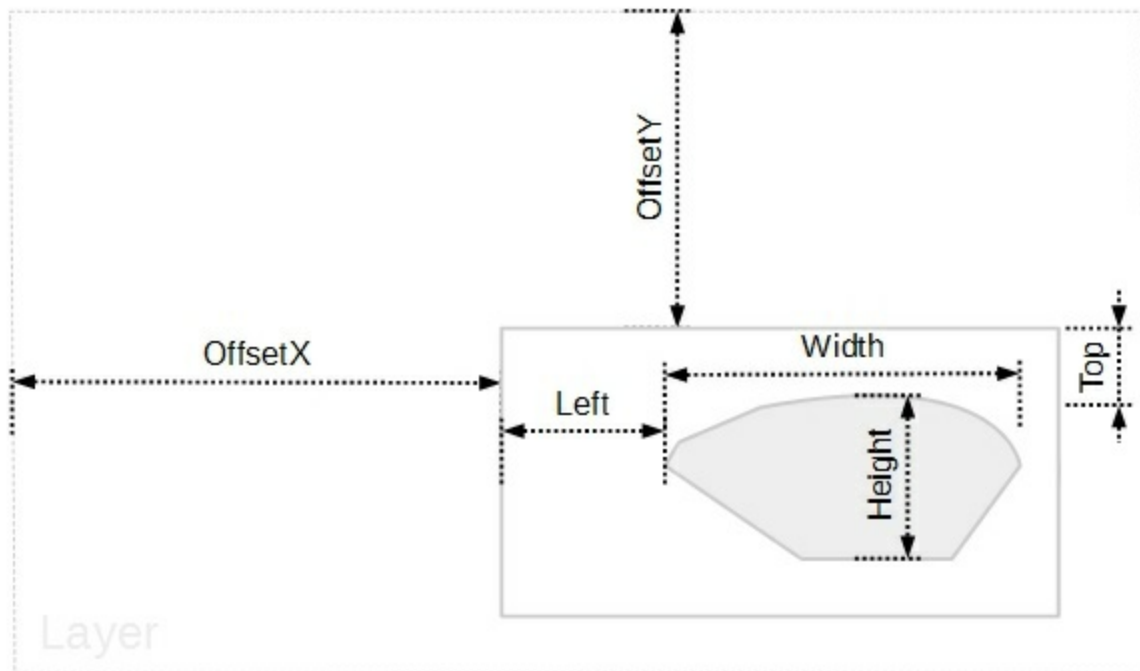
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:





The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipPicture.Height as String

Specifies the height value / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the height value / expression of the clip, relative to the layer.

By default, the Height property is empty, which indicates the height of the layer. The Height property is "height", if the expression is missing or invalid.

For instance:

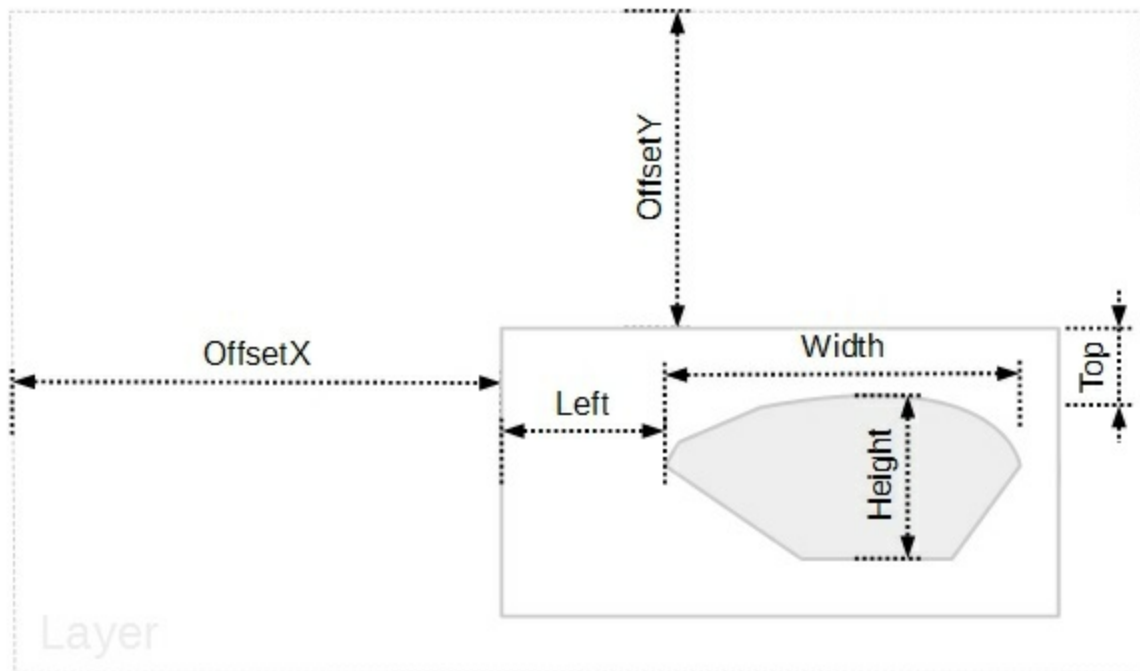
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPicture.InverseClip as Boolean

Indicates whether the current clip object is inverted.

Type	Description
Boolean	A Boolean expression that indicates whether the current clip object is inverted.

By default, InverseClip property is False. The InverseClip property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

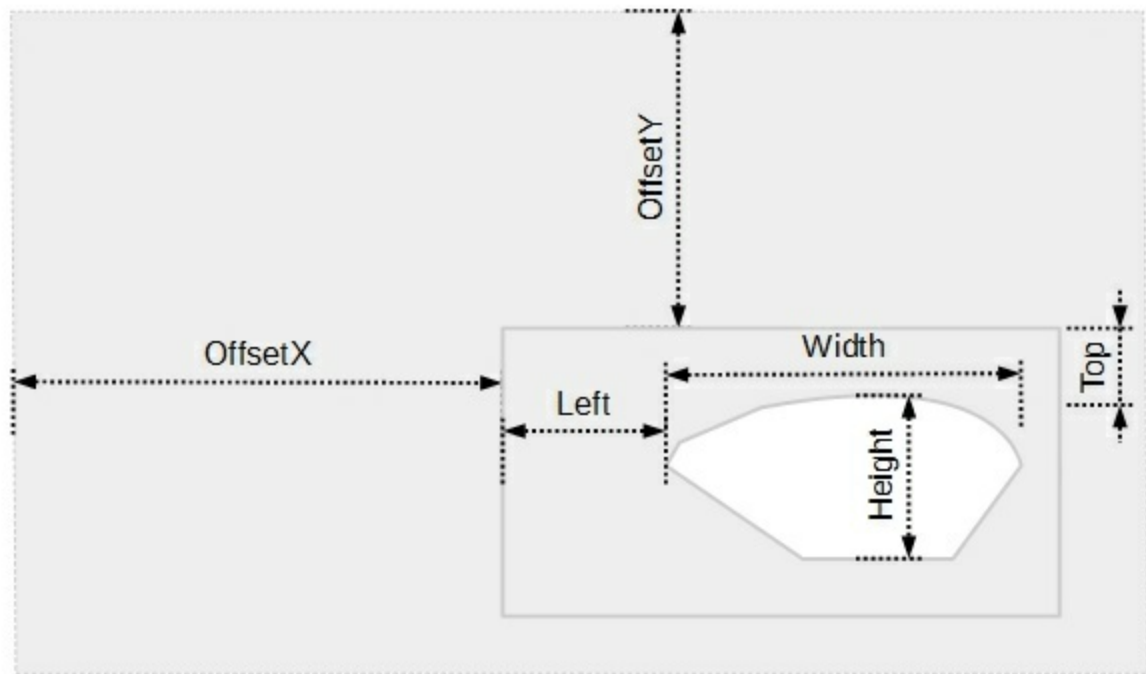
- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- Height, Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:



## property ClipPicture.Left as String

Specifies the left position / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the left position / expression of the clip, relative to the layer.

By default, the Left property is empty, which indicates the value of 0 ( left side of the layer ). The Left property is 0, if the expression is missing or invalid.

For instance:

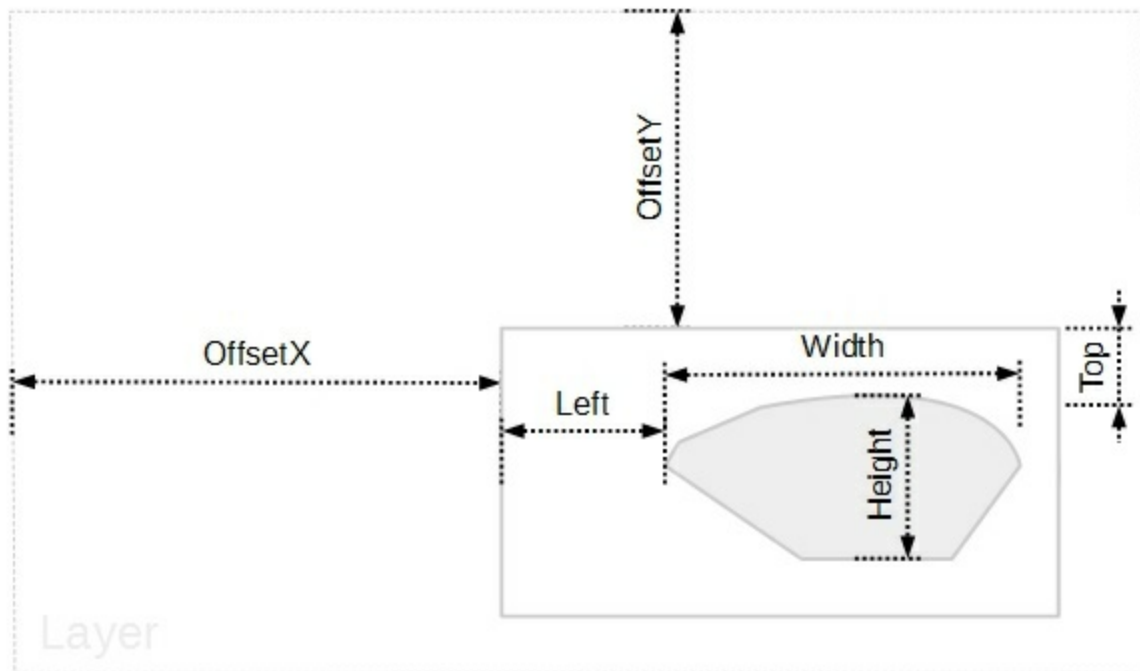
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPicture.Name as Variant

Indicates the picture to be used as a mask/clip.

Type	Description
Variant	<p>The Name property could be one of the following:</p> <ul style="list-style-type: none"><li>• A String expression indicates:<ul style="list-style-type: none"><li>◦ a name of a picture file in the PicturePath folder. For instance, Name = "Layer1.png", loads the Layer1.png file if found in the <a href="#">PicturesPath</a> folder.</li><li>◦ a picture file including its absolute path. For instance, Name = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\K loads the Layer1.png file from absolute path</li><li>◦ a key of the HTML picture, previously loaded by the HTMLPicture method. For instance, Name = "pic1", loads the HTML picture with the key pic1, so the pic1 should be load previously with a HTMLPicture call like HTMLPicture("pic1") = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\K</li><li>◦ an encode BASE64 string of a picture file. The Exontrol's <a href="#">ExImages</a> Tool encode/decode BASE64 strings from/to pictures. In this case, the string starts with "gB..", "gC.." and so on.</li></ul></li><li>• A Picture object that indicates the picture to be displayed. For instance, Name = LoadPicture("picture.jpg")</li></ul>

By default, the Name property is empty, so no clipping is applied. The picture being used as a clipping region must support transparency / alpha blending ( picture's attribute includes the PICTURE\_TRANSPARENT ). For instance, you can use any PNG file with transparency. By default, the opaque-pixels in the picture defines the clipping region. Use the [AlphaFrom](#) / [AlphaTo](#) to include semi-transparent pixels in the clipping region. The [DisplayAs](#) property retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.

To define a picture clip region over the layer you can use any of the following properties:

- Name, Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is

arranged on the mask/clip.

- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

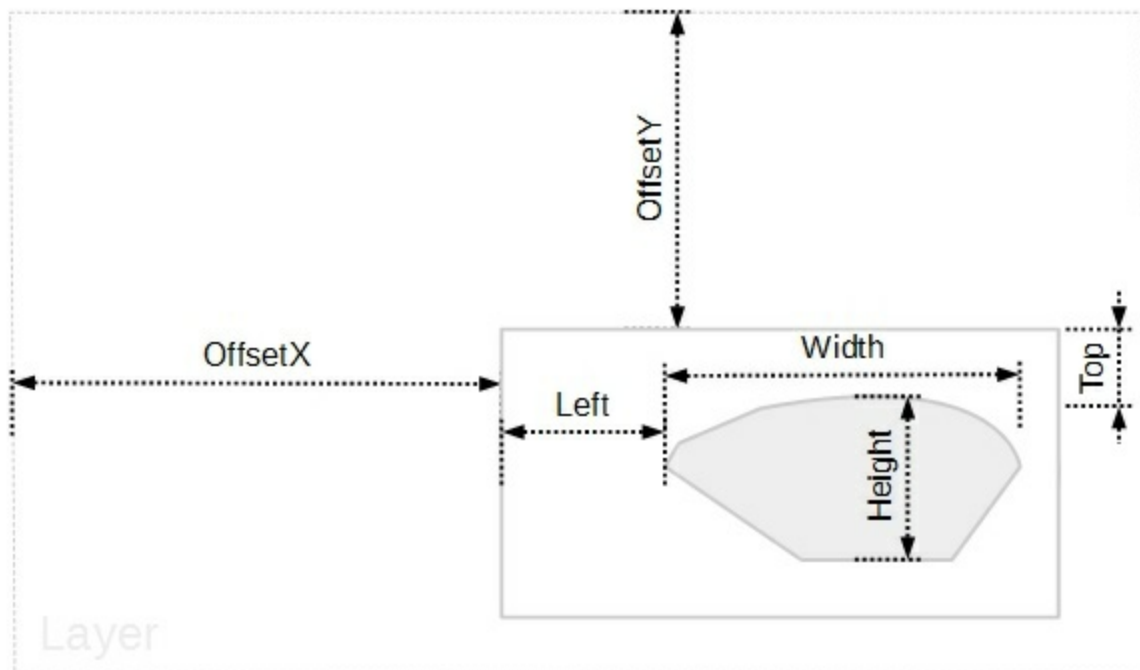
- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- Height, Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:



The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipPicture.OffsetX as String

Specifies the x-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-offset expression / value of the clip, relative to the layer.

By default, the OffsetX property is empty, which indicates the value of 0 ( left side of the layer ). The OffsetX property is 0, if the expression is missing or invalid.

For instance:

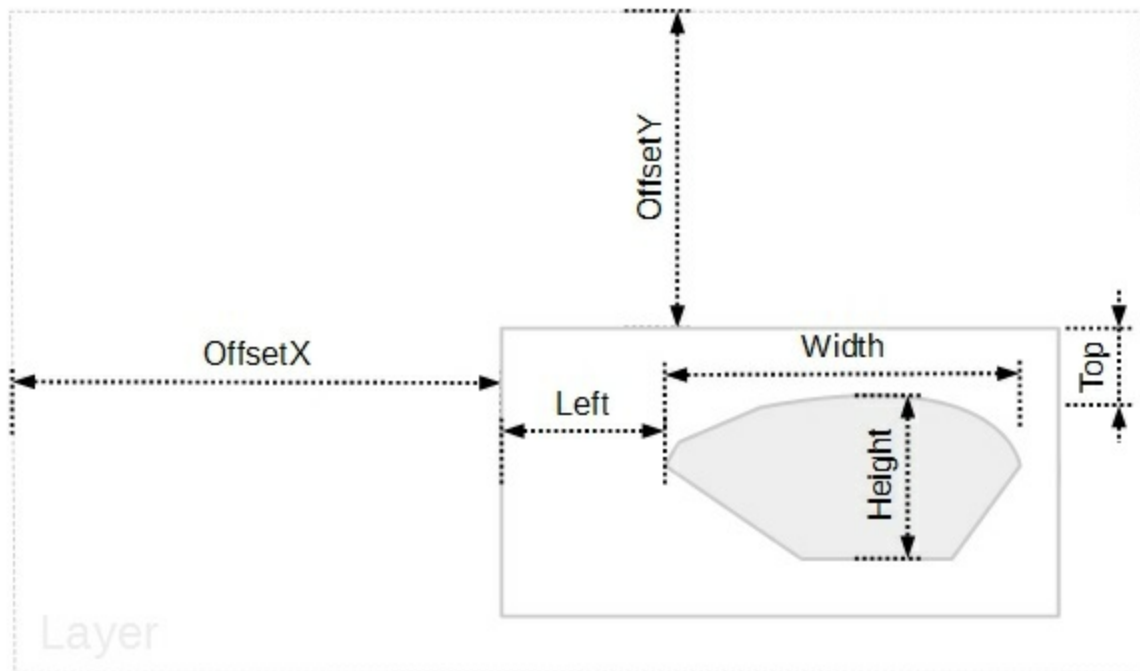
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPicture.OffsetY as String

Specifies the y-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-offset expression / value of the clip, relative to the layer.

By default, the OffsetY property is empty, which indicates the value of 0 ( top side of the layer ). The OffsetY property is 0, if the expression is missing or invalid.

For instance:

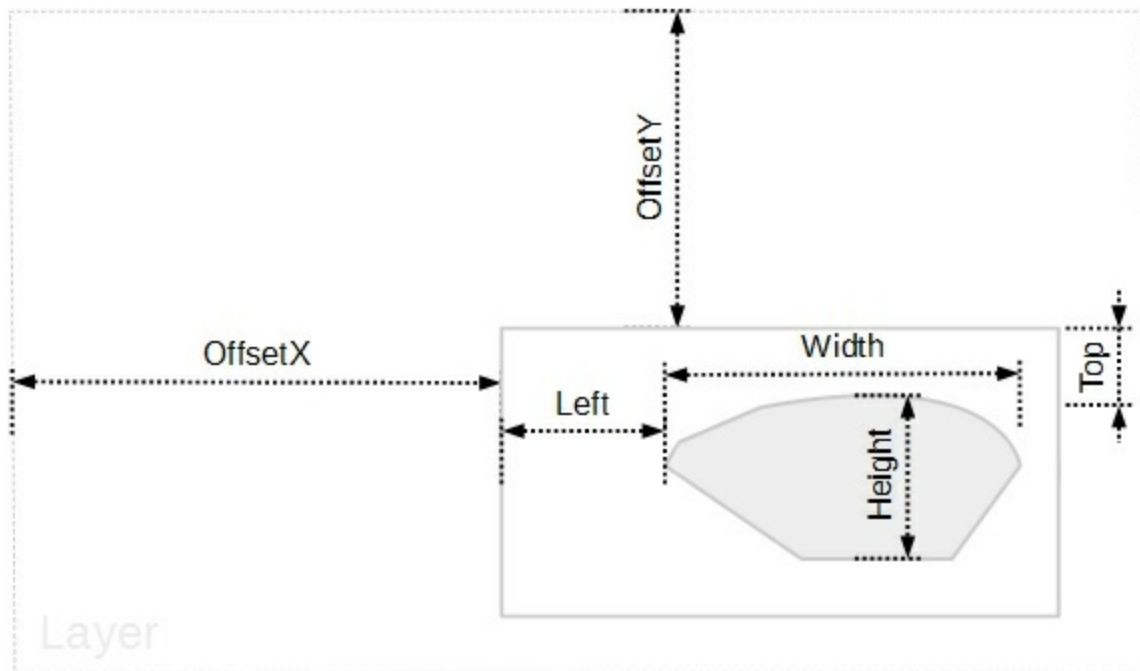
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipPicture.Top as String

Specifies the top position / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the top position / expression of the clip, relative to the layer.

By default, the Top property is empty, which indicates the value of 0 ( top side of the layer ). The Top property is 0, if the expression is missing or invalid.

For instance:

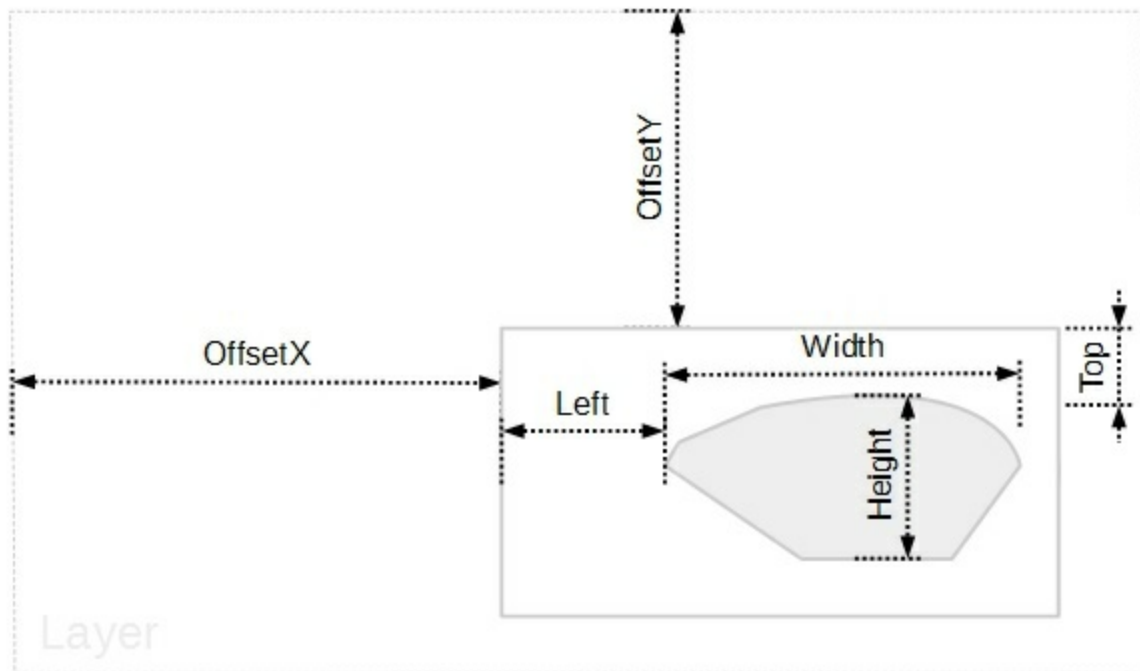
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPicture.Width as String

Specifies the width value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that specifies the width value / expression of the clip, relative to the layer.

By default, the Width property is empty, which indicates the width of the layer . The Width property is "width", if the expression is missing or invalid.

For instance:

- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

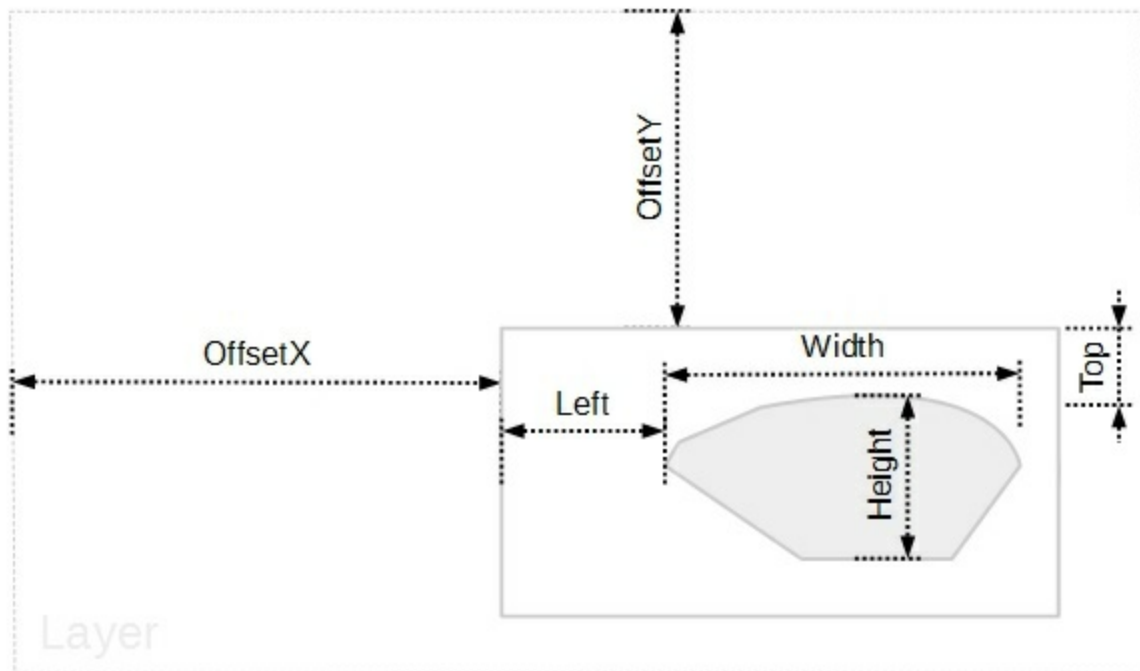
This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:





To define a picture clip region over the layer you can use any of the following properties:

- [Name](#), Indicates the picture to be used as a mask/clip.
- [DisplayAs](#), Retrieves or sets a value that indicates the way how the graphic is arranged on the mask/clip.
- [AlphaFrom](#), Gets or sets a value that specifies the alpha-byte to start clipping the picture from.
- [AlphaTo](#), Gets or sets a value that specifies the alpha-byte to end clipping the picture to.

To specify the position / size of the picture clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

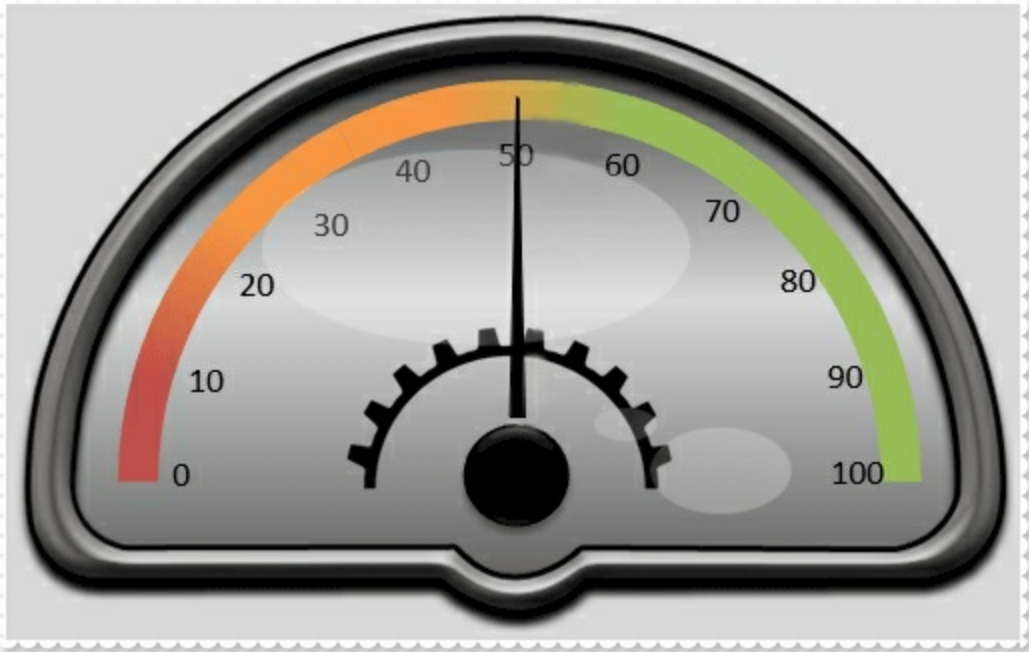
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

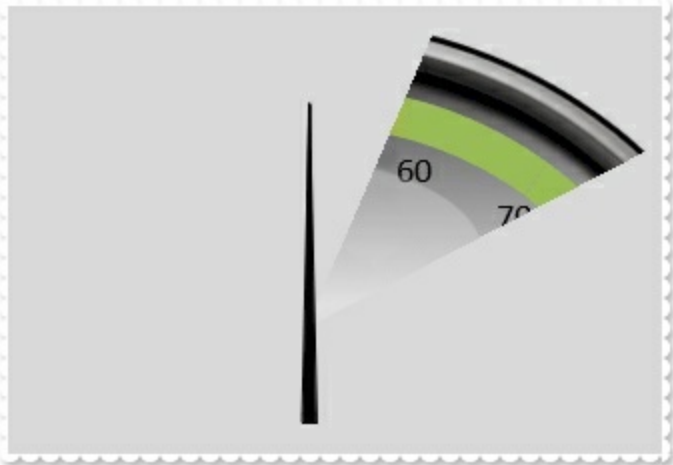
# ClipPie object

The ClipPie object holds information about a pie clip region.

For instance, having the following gauge:



an pie clip region over the background layer shows as:



Name	Description
<a href="#">CenterX</a>	Specifies the x-position / expression of the center of the clip, relative to the layer.
<a href="#">CenterY</a>	Specifies the y-position / expression of the center of the clip, relative to the layer.
<a href="#">InverseClip</a>	Indicates whether the current clip object is inverted.
	Specifies the x-offset expression / value of the clip,

<a href="#">OffsetX</a>	relative to the layer.
<a href="#">OffsetY</a>	Specifies the y-offset expression / value of the clip, relative to the layer.
<a href="#">RadiusX</a>	Specifies the x-radius value / expression of the clip, relative to the layer.
<a href="#">RadiusY</a>	Specifies the y-radius value / expression of the clip, relative to the layer.
<a href="#">StartAngle</a>	Specifies the starting angle in degrees relative to the y-axis.
<a href="#">SweepAngle</a>	Specifies the sweep angle in degrees relative to the starting angle.

## property ClipPie.CenterX as String

Specifies the x-position / expression of the center of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-position / expression of the center of the clip, relative to the layer.

By default, the CenterX property is empty, which indicates the center of the layer. The CenterX property is "width/2" ( center of the layer ), if the expression is missing or invalid.

For instance:

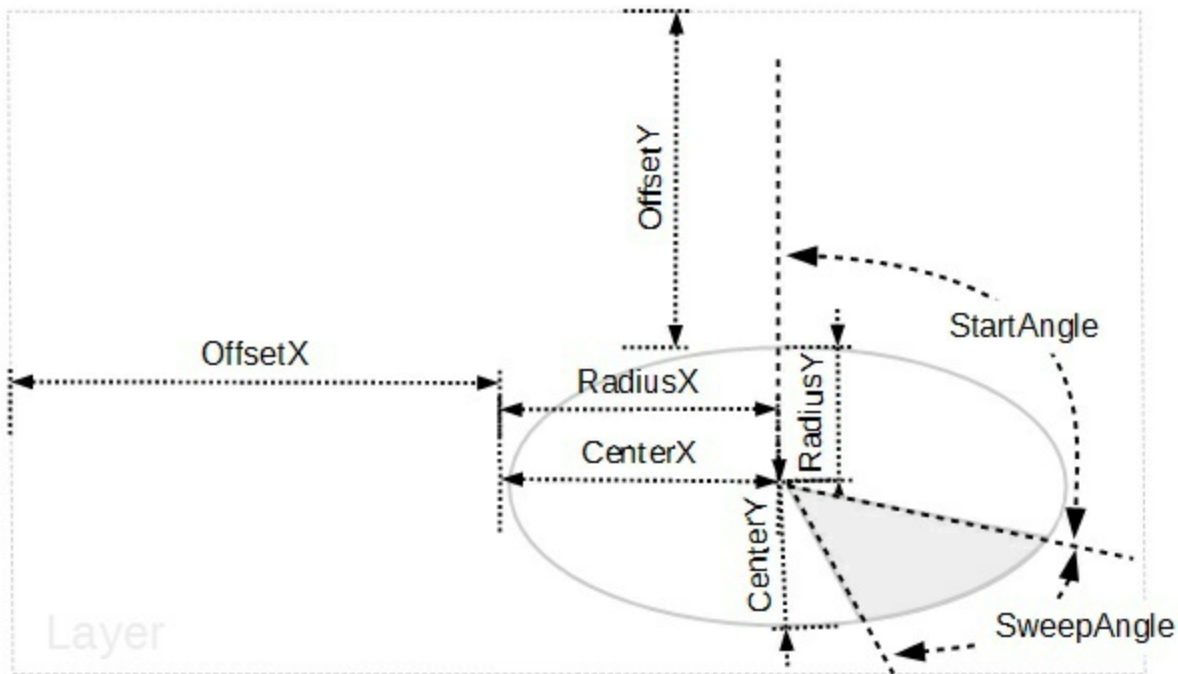
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- **CenterX**, Specifies the x-position / expression of the center of the clip, relative to the layer.
- **CenterY**, Specifies the y-position / expression of the center of the clip, relative to the layer.
- **RadiusX**, Specifies the x-radius value / expression of the clip, relative to the layer.
- **RadiusY**, Specifies the y-radius value / expression of the clip, relative to the layer.
- **StartAngle**, Specifies the starting angle in degrees relative to the y-axis.
- **SweepAngle**, Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- **OffsetX**, Specifies the x-offset expression / value of the clip, relative to the layer.
- **OffsetY**, Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The **InverseClip** property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPie.CenterY as String

Specifies the y-position / expression of the center of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-position / expression of the center of the clip, relative to the layer.

By default, the CenterY property is empty, which indicates the center of the layer The CenterY property is "height/2" ( center of the layer ), if the expression is missing or invalid.

For instance:

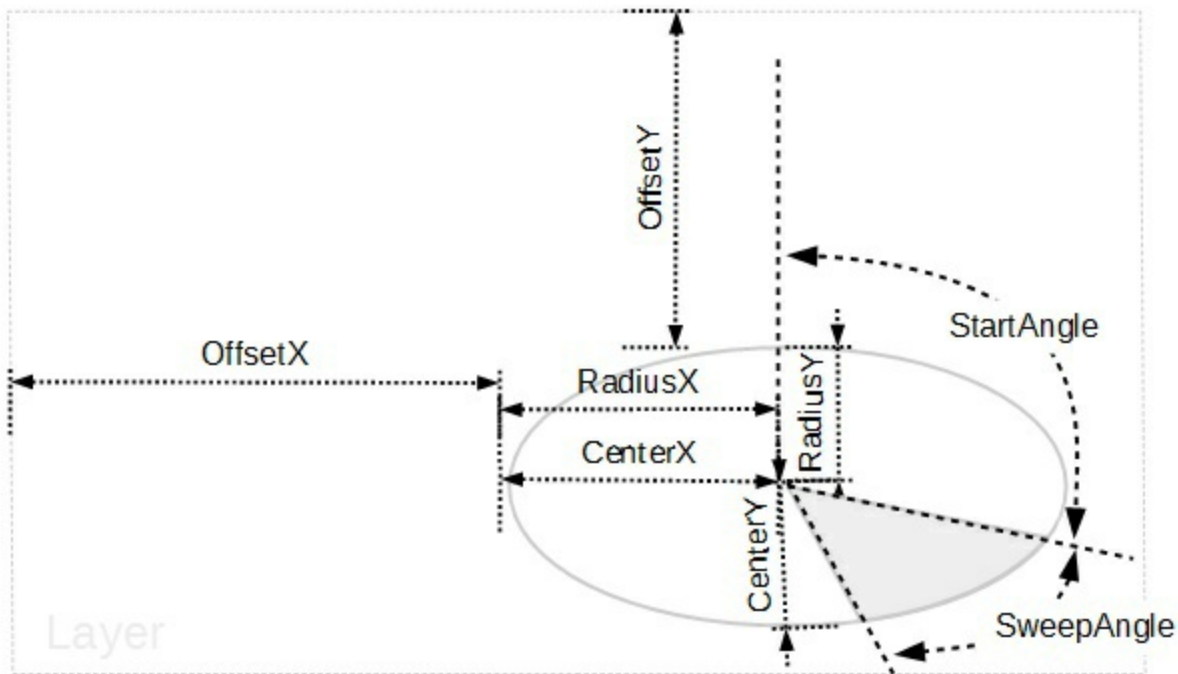
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or right side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

## property ClipPie.InverseClip as Boolean

Indicates whether the current clip object is inverted.

Type	Description
Boolean	A Boolean expression that indicates whether the current clip object is inverted.

By default, InverseClip property is False. The InverseClip property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

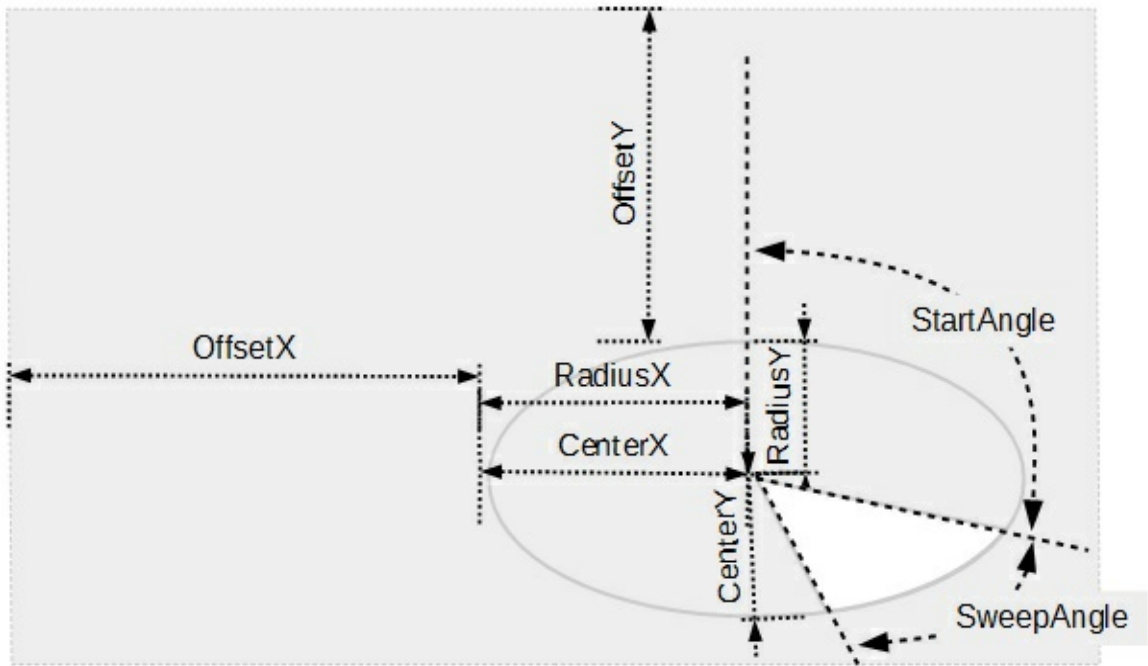
To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:





# property ClipPie.OffsetX as String

Specifies the x-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-offset expression / value of the clip, relative to the layer.

By default, the OffsetX property is empty, which indicates the value of 0 ( left side of the layer ). The OffsetX property is 0, if the expression is missing or invalid.

For instance:

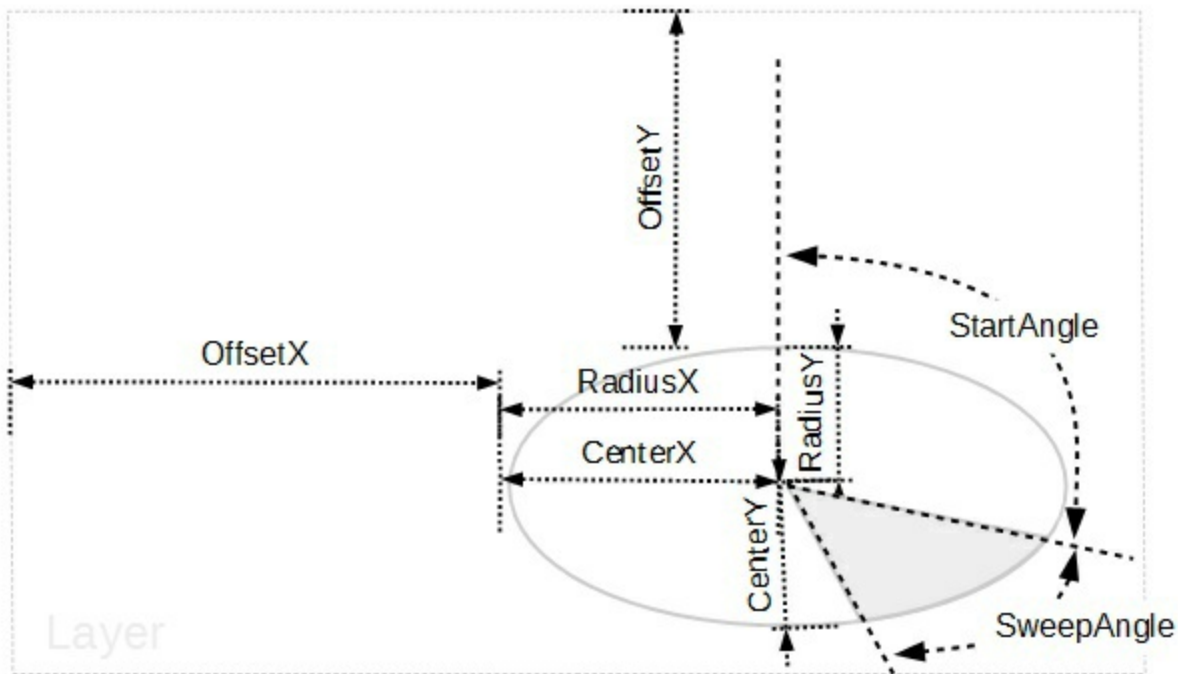
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPie.OffsetY as String

Specifies the y-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-offset expression / value of the clip, relative to the layer.

By default, the OffsetY property is empty, which indicates the value of 0 ( top side of the layer ). The OffsetY property is 0, if the expression is missing or invalid.

For instance:

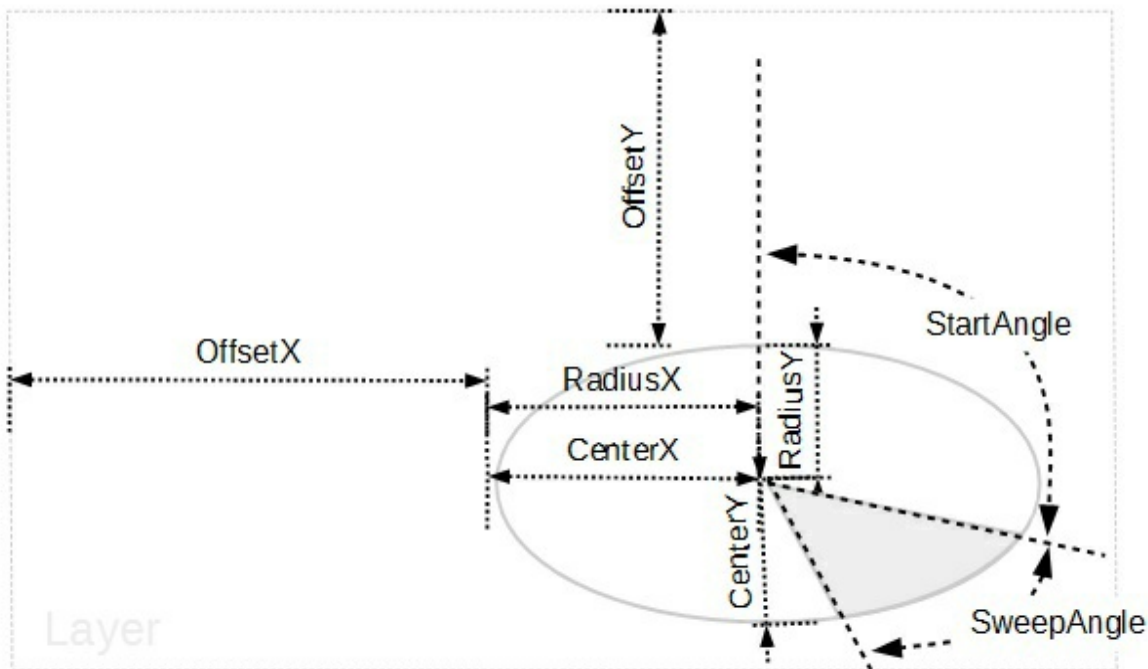
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPie.RadiusX as String

Specifies the x-radius value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-radius value / expression of the clip, relative to the layer.

By default, the RadiusX property is empty, which indicates the half of the layer's width. The RadiusX property is "width/2" ( half of the layer's width ), if the expression is missing or invalid.

For instance:

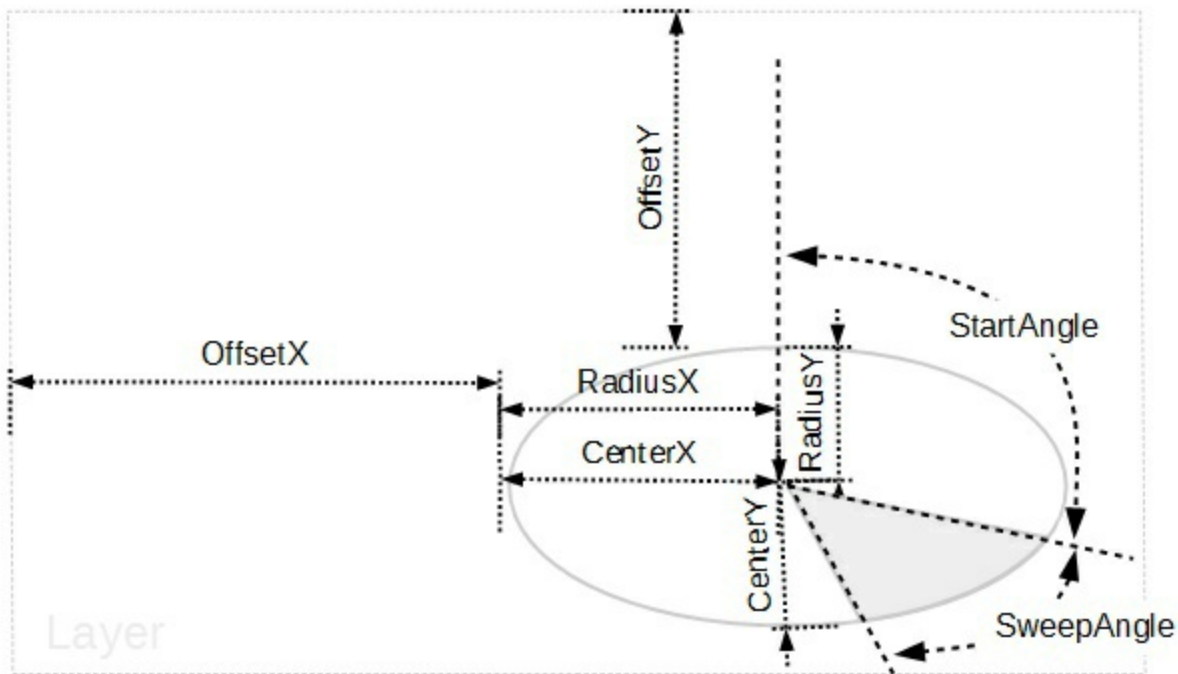
- "15", 15 pixels radius
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPie.RadiusY as String

Specifies the y-radius value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-radius value / expression of the clip, relative to the layer.

By default, the RadiusY property is empty, which indicates the half of the layer's height. The RadiusY property is "height/2" ( half of the layer's height ), if the expression is missing or invalid.

For instance:

- "15", 15 pixels radius
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or right side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

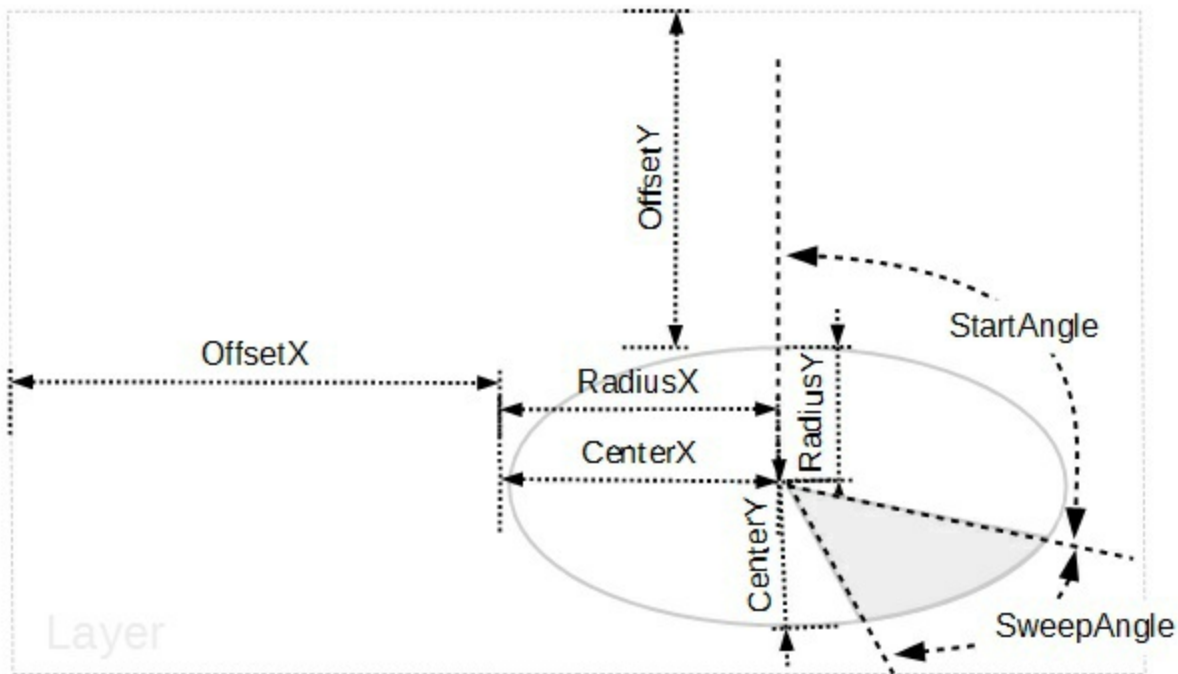
This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:





To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPie.StartAngle as String

Specifies the starting angle in degrees relative to the y-axis.

Type	Description
String	A String expression that defines the starting angle in degrees relative to the y-axis.

By default, the StartAngle property is empty, which indicates 0 degree. The StartAngle property is 0 degree, if empty, missing or invalid.

For instance:

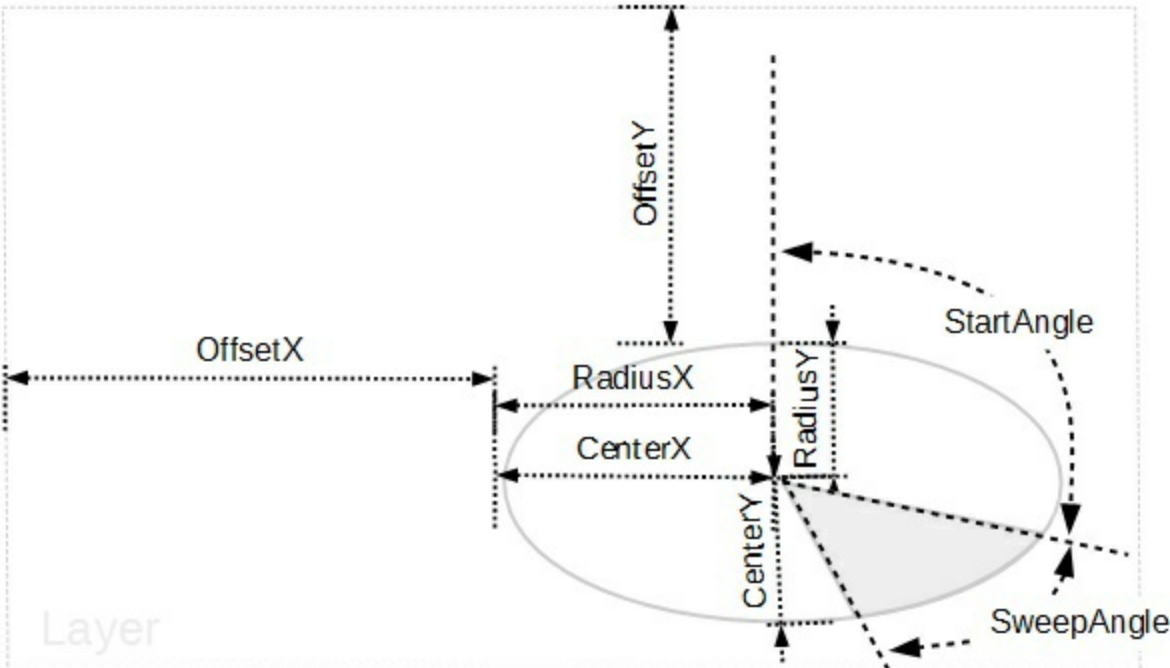
- "-45", 45 degree anti-clockwise
- "90", 90 degree clockwise
- "value" specifies that the angle is equal with the clip's [Value](#) property
- "value / 100 \* 360" results the percent [Value](#) from 360 degree

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPie.SweepAngle as String

Specifies the sweep angle in degrees relative to the starting angle.

Type	Description
String	A String expression that defines the sweep angle in degrees relative to the starting angle.

By default, the SweepAngle property is empty, which indicates 0 degree. The SweepAngle property is 0 degree, if empty, missing or invalid.

For instance:

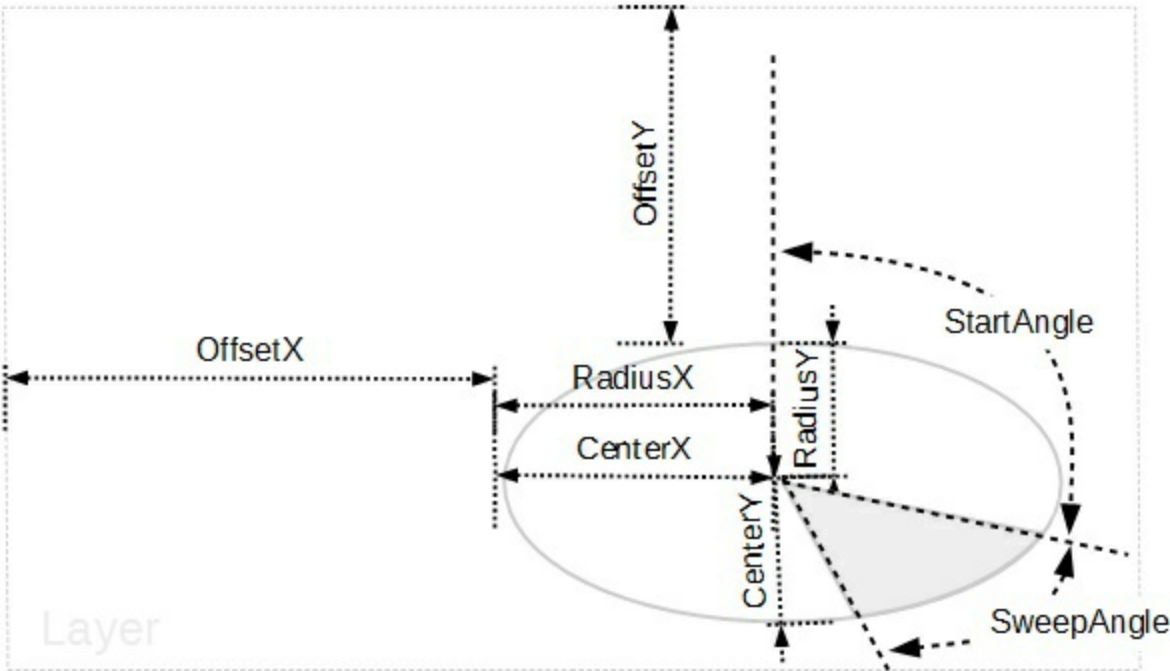
- "-45", 45 degree anti-clockwise
- "90", 90 degree clockwise
- "value" specifies that the angle is equal with the clip's [Value](#) property
- "value / 100 \* 360" results the percent [Value](#) from 360 degree

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an elliptical clip region over the layer you can use any of the following properties:

- [CenterX](#), Specifies the x-position / expression of the center of the clip, relative to the layer.
- [CenterY](#), Specifies the y-position / expression of the center of the clip, relative to the layer.
- [RadiusX](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [RadiusY](#), Specifies the y-radius value / expression of the clip, relative to the layer.
- [StartAngle](#), Specifies the starting angle in degrees relative to the y-axis.
- [SweepAngle](#), Specifies the sweep angle in degrees relative to the starting angle.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

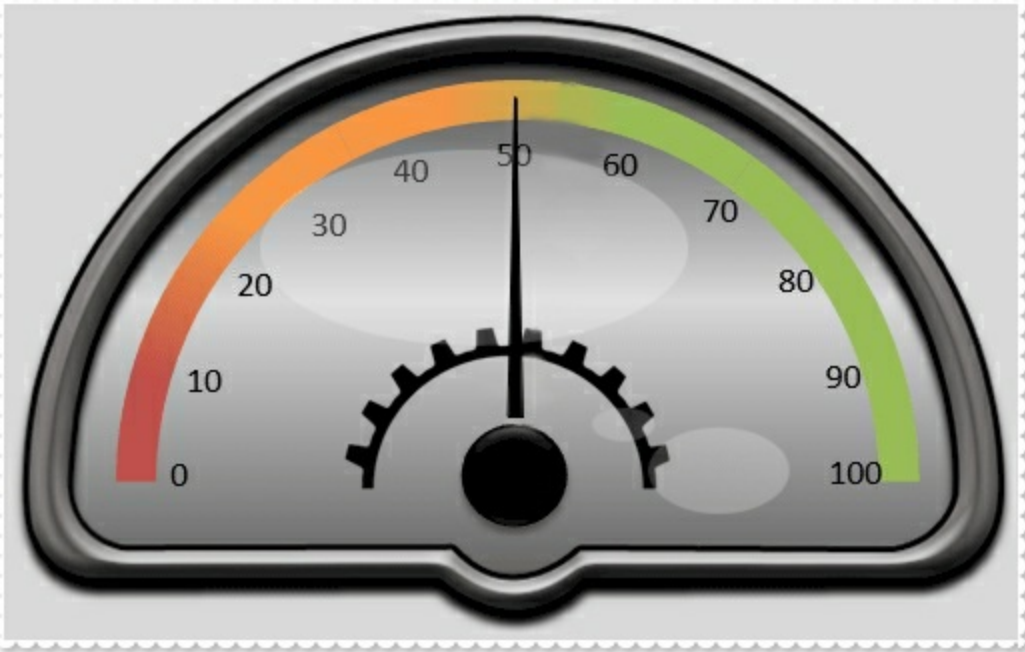
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

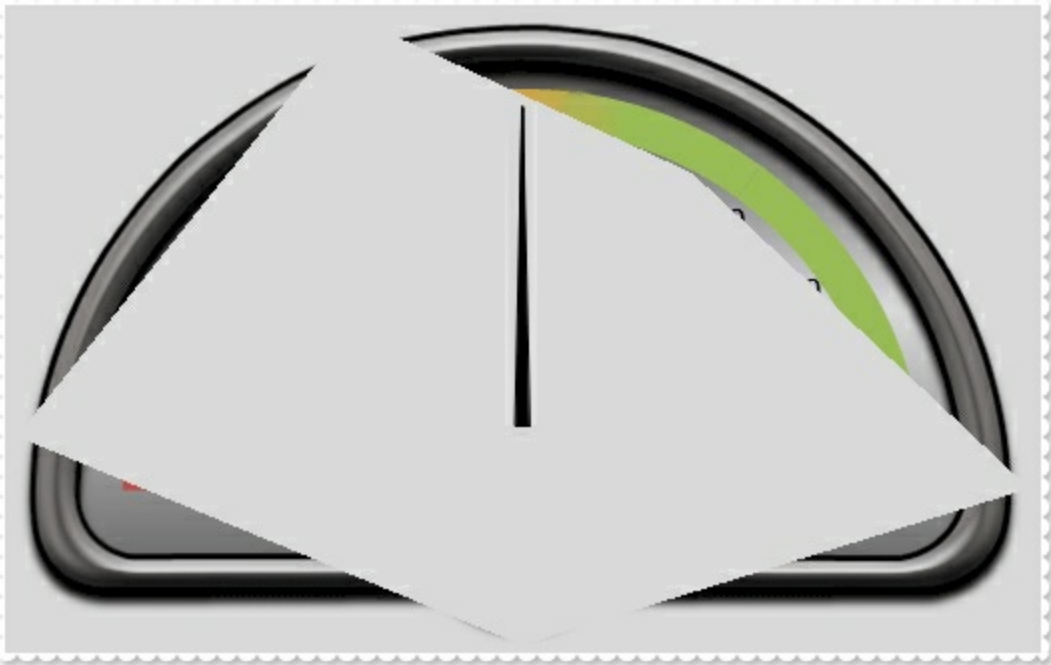
# ClipPolygon object

The ClipPolygon object holds information about polygonal clip region.

For instance, having the following gauge:



an polygonal clip region over the background layer shows as:



The ClipPolygon object supports the following properties and methods:

Name	Description
<a href="#">InverseClip</a>	Indicates whether the current clip object is inverted.
	Specifies the x-offset expression / value of the clip,

[OffsetX](#)

relative to the layer.

[OffsetY](#)

Specifies the y-offset expression / value of the clip, relative to the layer.

[Points](#)

Indicates the number of points that defines the polygon.

[X](#)

Specifies the x-position expression / value of the point, relative to the layer.

[Y](#)

Specifies the y-position expression / value of the point, relative to the layer.

# property ClipPolygon.InverseClip as Boolean

Indicates whether the current clip object is inverted.

Type	Description
Boolean	A Boolean expression that indicates whether the current clip object is inverted.

By default, InverseClip property is False. The InverseClip property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

To define an polygonal clip region over the layer you can use any of the following properties:

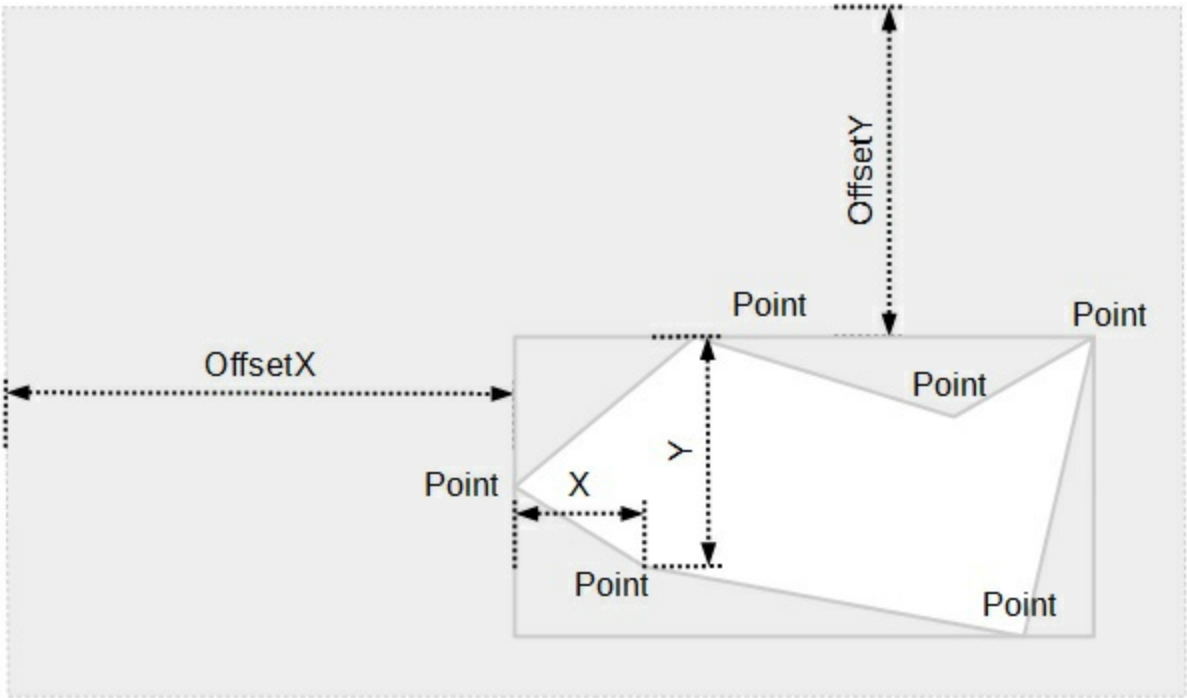
- [Points](#), Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [Y](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:





# property ClipPolygon.OffsetX as String

Specifies the x-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-offset expression / value of the clip, relative to the layer.

By default, the OffsetX property is empty, which indicates the value of 0 ( left side of the layer ). The OffsetX property is 0, if the expression is missing or invalid.

For instance:

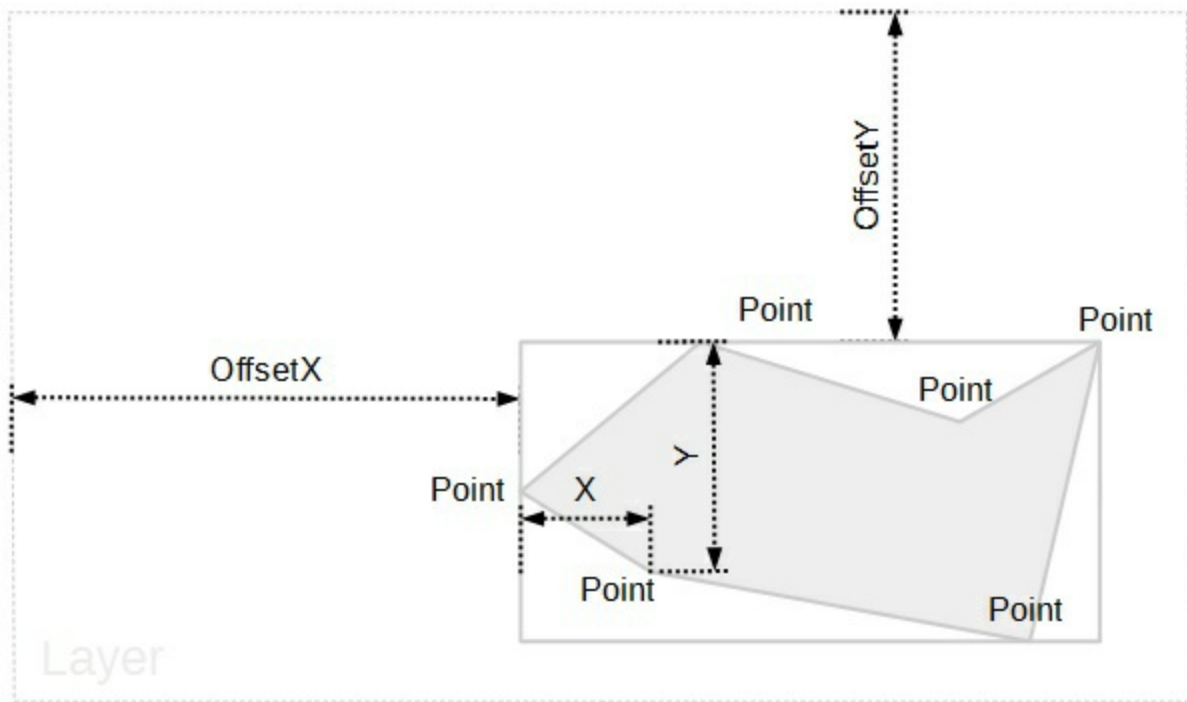
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an polygonal clip region over the layer you can use any of the following properties:

- [Points](#), Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [Y](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipPolygon.OffsetY as String

Specifies the y-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-offset expression / value of the clip, relative to the layer.

By default, the OffsetY property is empty, which indicates the value of 0 ( top side of the layer ). The OffsetY property is 0, if the expression is missing or invalid.

For instance:

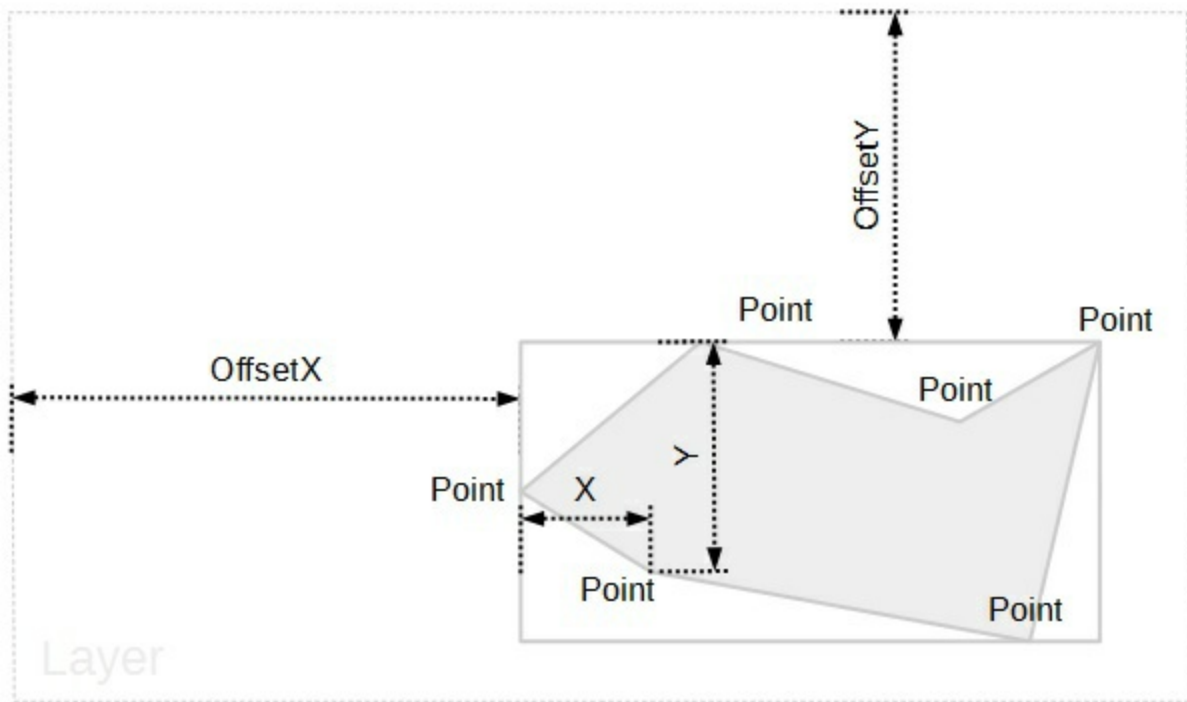
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an polygonal clip region over the layer you can use any of the following properties:

- [Points](#), Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [Y](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

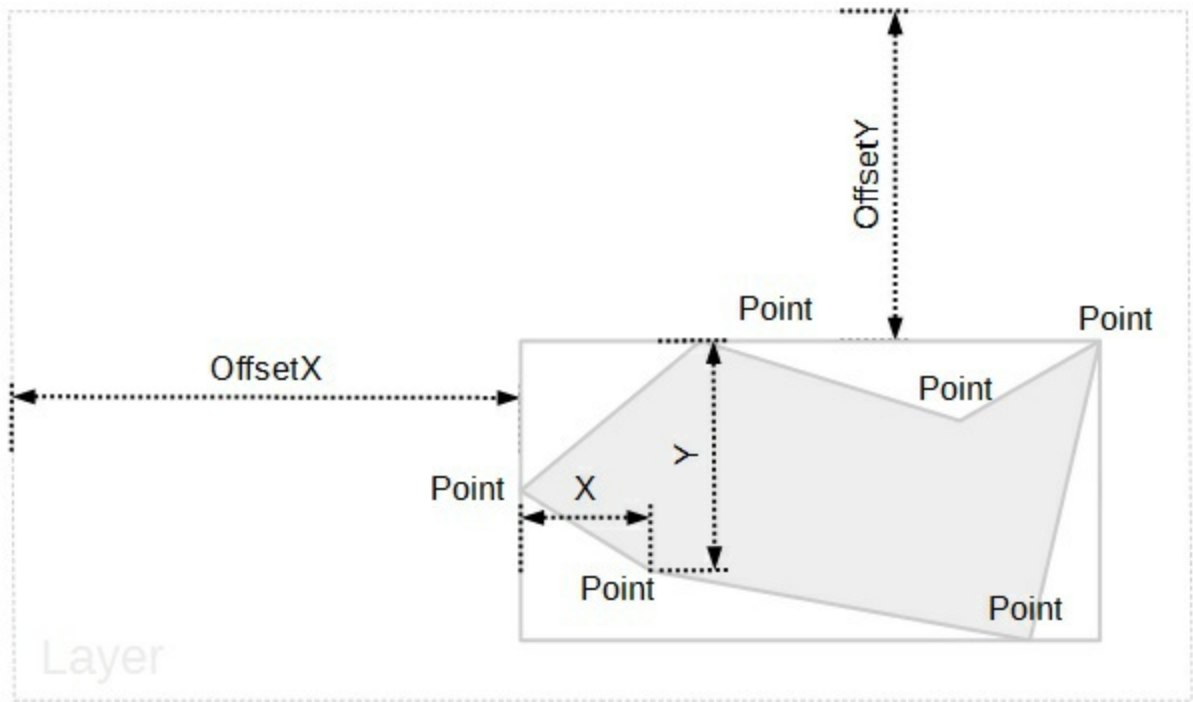
# property ClipPolygon.Points as Long

Indicates the number of points that defines the polygon.

Type	Description
Long	A Long expression that defines the number of points within the polygon. The number of points must be greater or equal with 3.

By default, the Points property is 0. The Points property indicates the number of points that defines the polygon. The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

The following screen shot shows properties of the clipping objects relative to the layer:



To define an polygonal clip region over the layer you can use any of the following properties:

- Points, Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- Y, Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.



# property ClipPolygon.X(Index as Long) as String

Specifies the x-position expression / value of the point, relative to the layer.

Type	Description
Index as Long	A Long expression that specifies the index of the point, whose x-coordinate is accessed. The Index parameter is zero-based, so it can be 0, 1, 2, ... <a href="#">Points</a> - 1
String	A String value that specifies the x-position expression / value of the point, relative to the layer.

By default, the X property is empty, which indicates the value of 0 ( left side of the layer ). The X property is 0, if the expression is empty, missing or invalid. The [Points](#) property indicates the number of points that defines the polygon.

For instance:

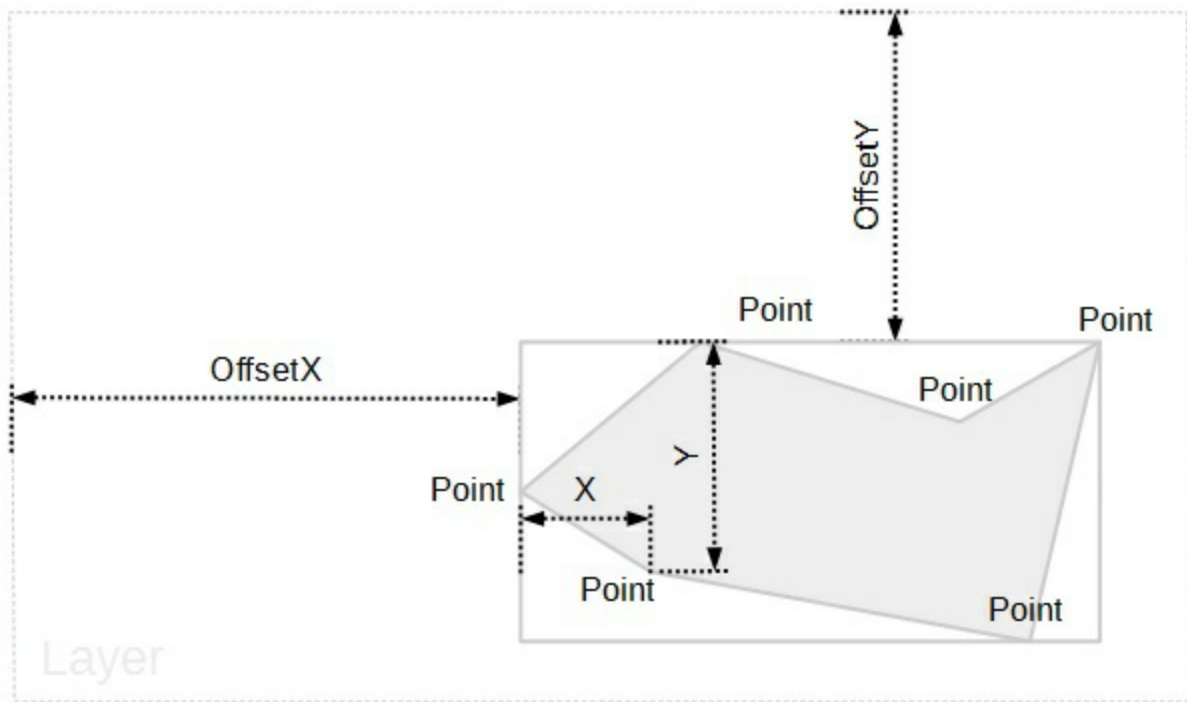
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an polygonal clip region over the layer you can use any of the following properties:

- [Points](#), Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [Y](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipPolygon.Y(Index as Long) as String

Specifies the y-position expression / value of the point, relative to the layer.

Type	Description
Index as Long	A Long expression that specifies the index of the point, whose x-coordinate is accessed. The Index parameter is zero-based, so it can be 0, 1, 2, ... <a href="#">Points</a> - 1
String	A String value that specifies the y-position expression / value of the point, relative to the layer.

By default, the Y property is empty, which indicates the value of 0 ( top side of the layer ). The Y property is 0, if the expression is empty, missing or invalid. The [Points](#) property indicates the number of points that defines the polygon.

For instance:

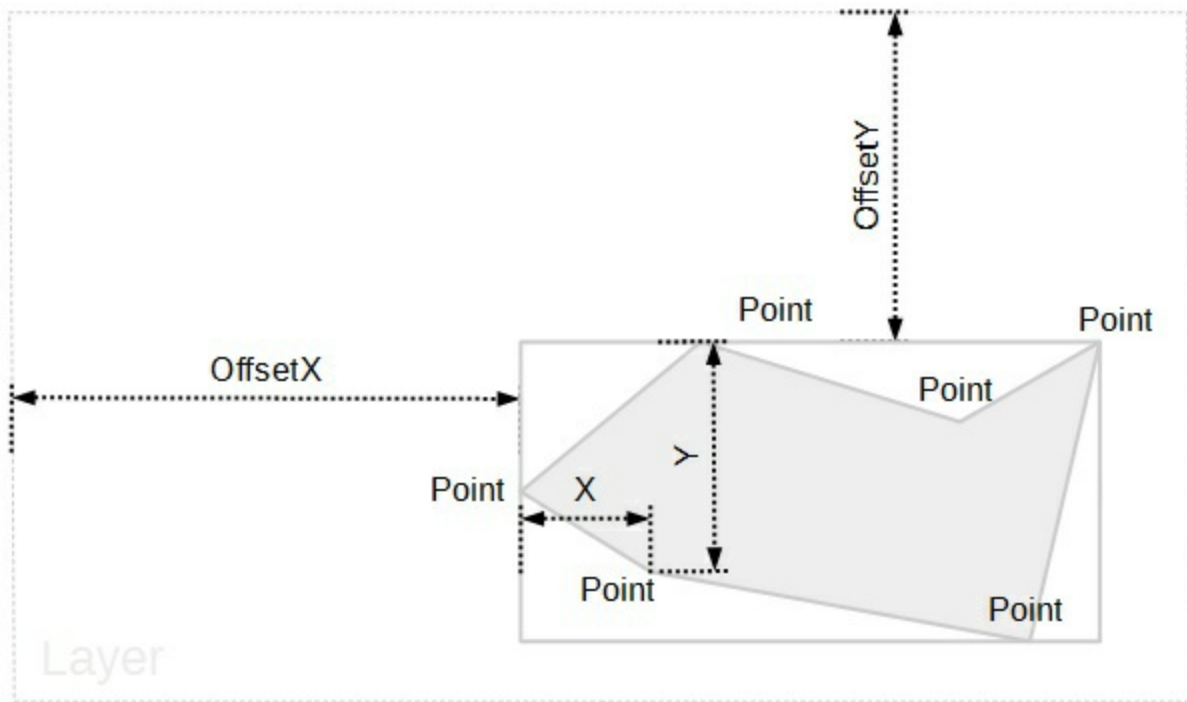
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define an polygonal clip region over the layer you can use any of the following properties:

- [Points](#), Indicates the number of points that defines the polygon.
- [X](#), Specifies the x-radius value / expression of the clip, relative to the layer.
- [Y](#), Specifies the y-radius value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

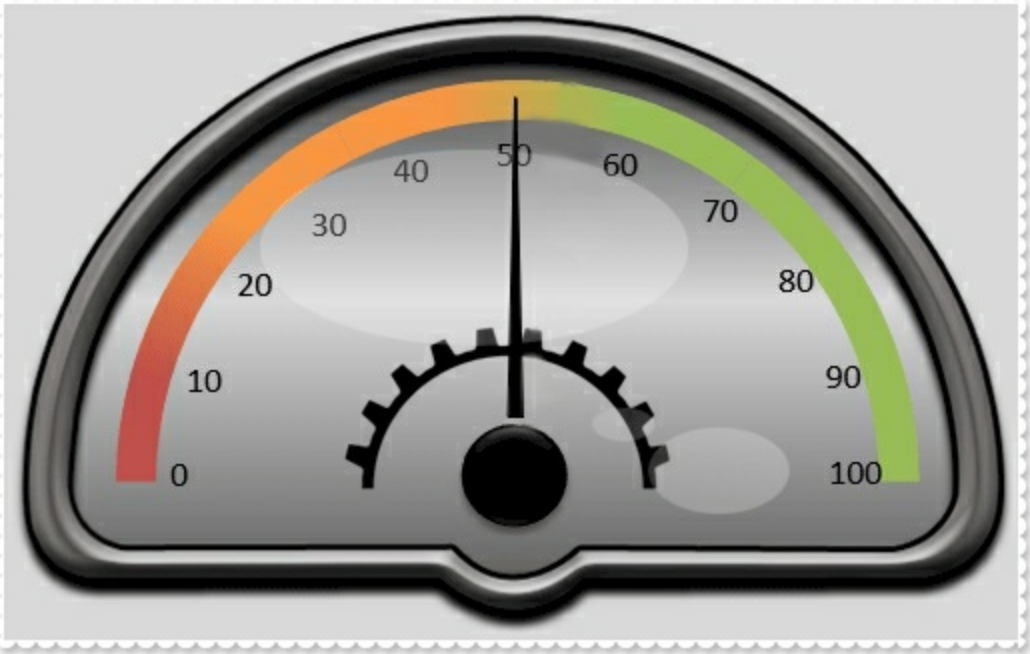
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

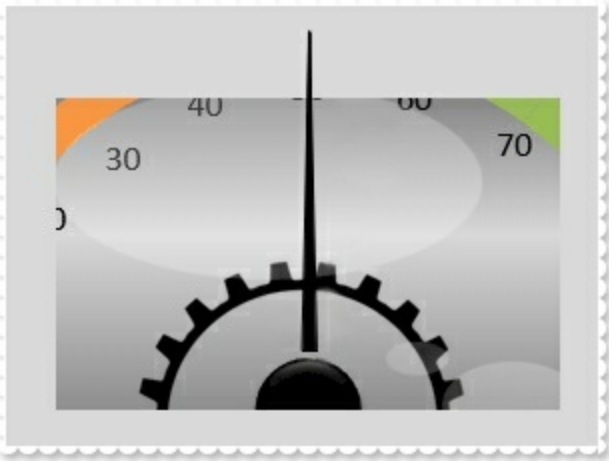
# ClipRectangle object

The ClipRectangle object holds information about an rectangular clip region.

For instance, having the following gauge:



a rectangular clip region over the background layer shows as:



The ClipRectangle property supports the following properties and methods:

Name	Description
<a href="#">Height</a>	Specifies the height value / expression of the clip, relative to the layer.
<a href="#">InverseClip</a>	Indicates whether the current clip object is inverted.
<a href="#">Left</a>	Specifies the left position / expression of the clip, relative to the layer.
<a href="#">OffsetX</a>	Specifies the x-offset expression / value of the clip,

relative to the layer.

[OffsetY](#)

Specifies the y-offset expression / value of the clip, relative to the layer.

[Top](#)

Specifies the top position / expression of the clip, relative to the layer.

[Width](#)

Specifies the width value / expression of the clip, relative to the layer.

# property ClipRectangle.Height as String

Specifies the height value / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the height value / expression of the clip, relative to the layer.

By default, the Height property is empty, which indicates the height of the layer. The Height property is "height", if the expression is missing or invalid.

For instance:

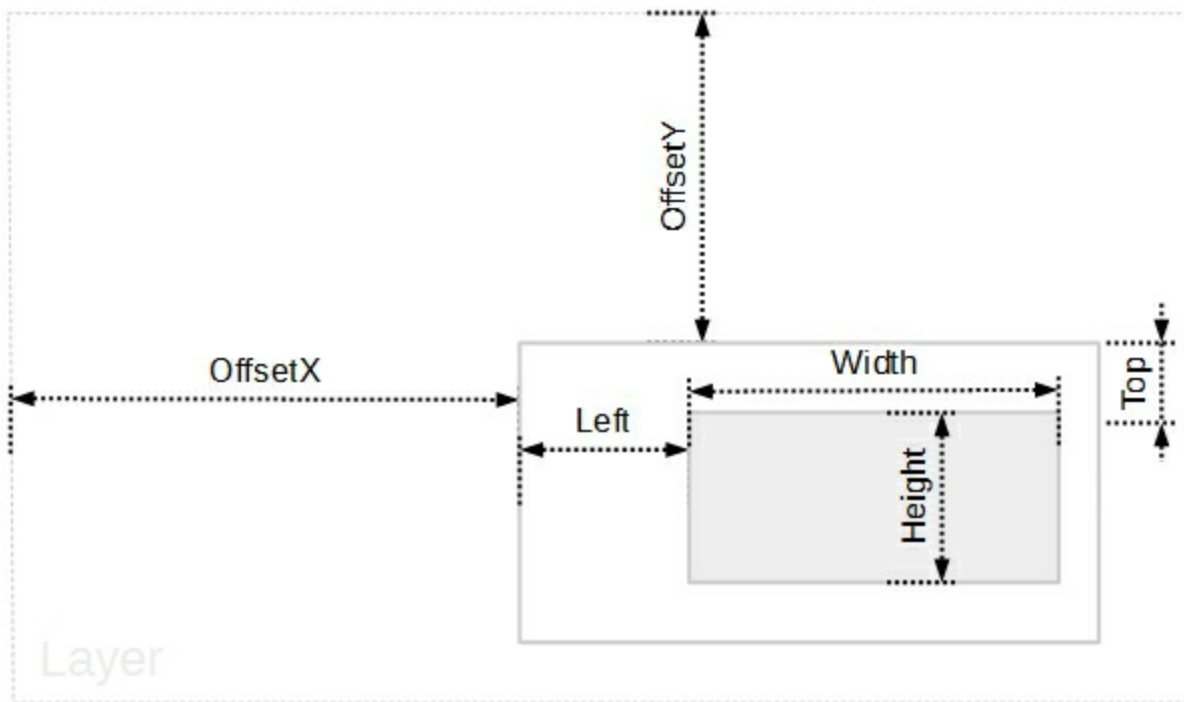
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRectangle.InverseClip as Boolean

Indicates whether the current clip object is inverted.

Type	Description
Boolean	A Boolean expression that indicates whether the current clip object is inverted.

By default, InverseClip property is False. The InverseClip property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

To define or specify the position / size of the rectangular clip object you can use any of the following properties:

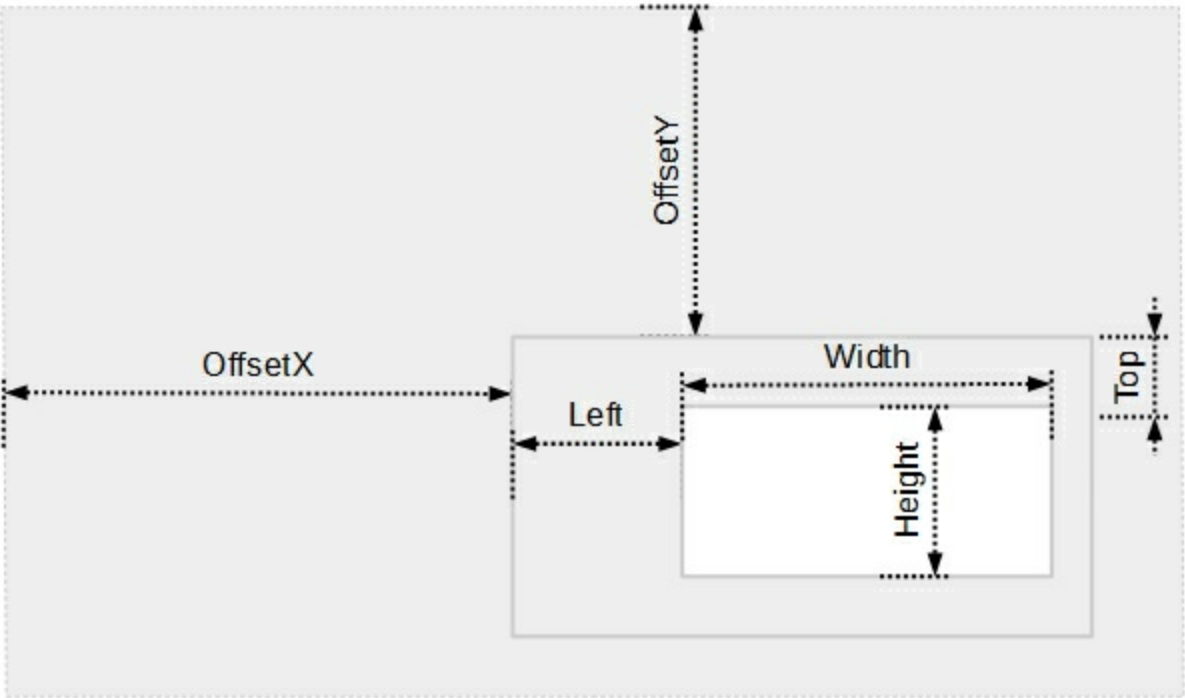
- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- Height, Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:







# property ClipRectangle.Left as String

Specifies the left position / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the left position / expression of the clip, relative to the layer.

By default, the Left property is empty, which indicates the value of 0 ( left side of the layer ). The Left property is 0, if the expression is missing or invalid.

For instance:

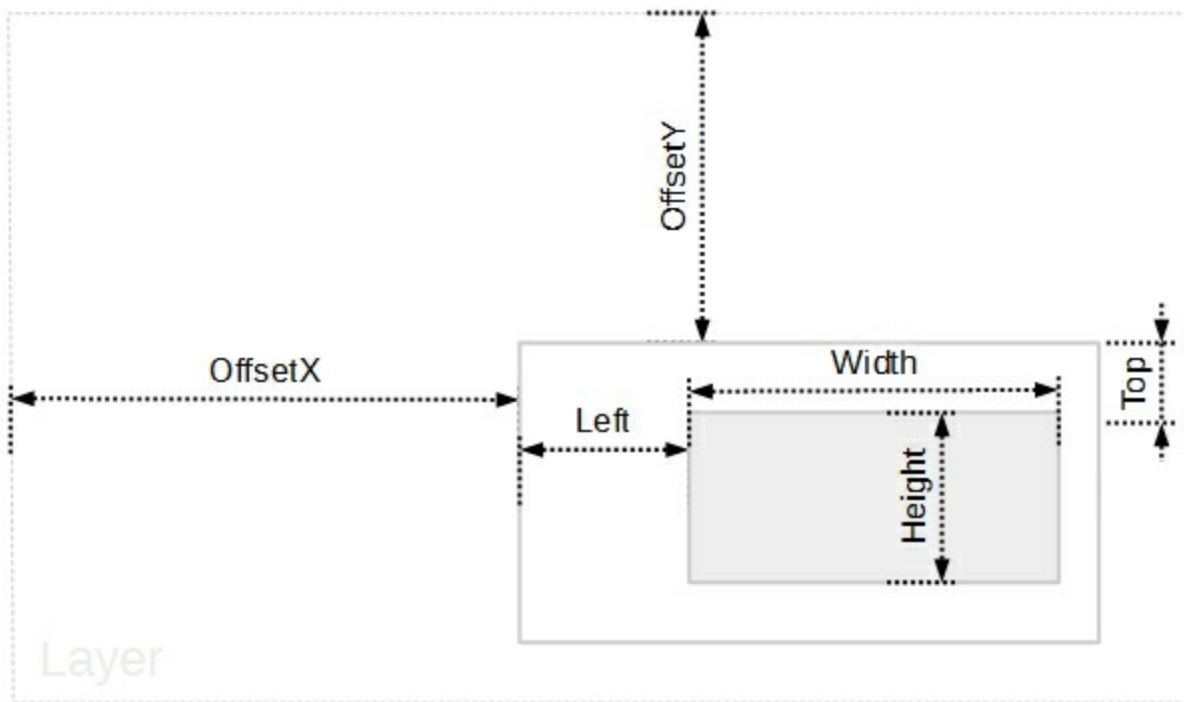
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- **Left**, Specifies the left position / expression of the clip, relative to the layer.
- **Top**, Specifies the top position / expression of the clip, relative to the layer.
- **Width**, Specifies the width value / expression of the clip, relative to the layer.
- **Height**, Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- **OffsetX**, Specifies the x-offset expression / value of the clip, relative to the layer.
- **OffsetY**, Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The **InverseClip** property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRectangle.OffsetX as String

Specifies the x-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-offset expression / value of the clip, relative to the layer.

By default, the OffsetX property is empty, which indicates the value of 0 ( left side of the layer ). The OffsetX property is 0, if the expression is missing or invalid.

For instance:

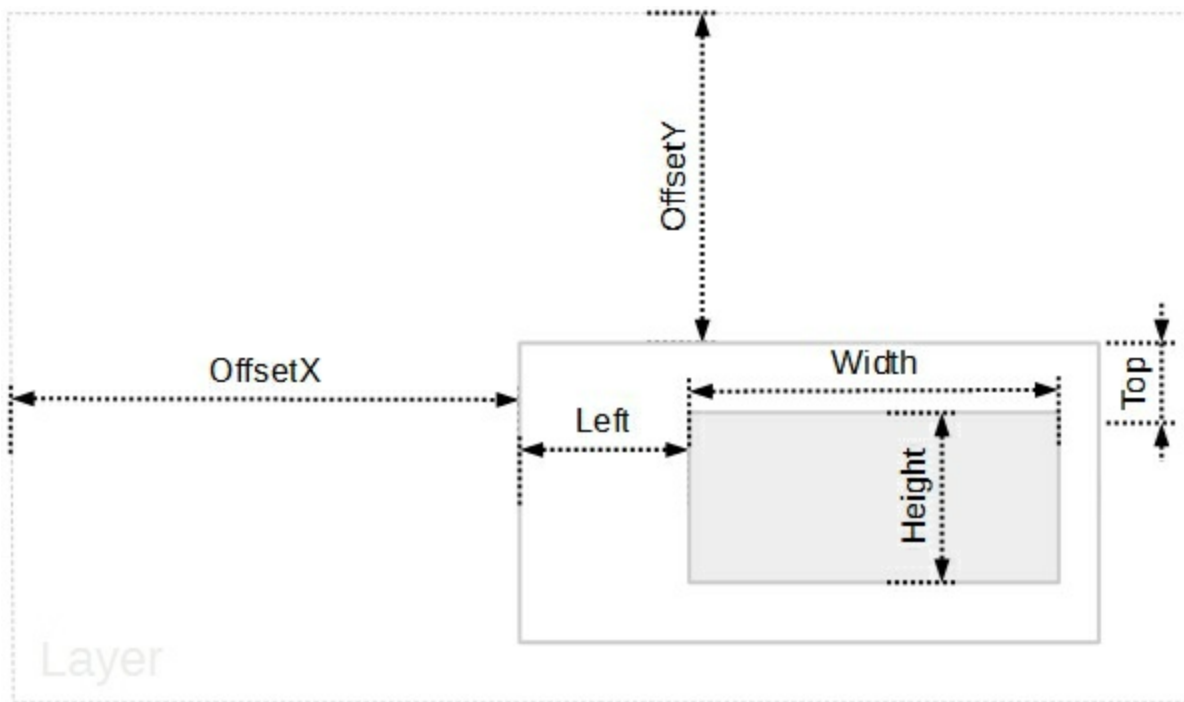
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRectangle.OffsetY as String

Specifies the y-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-offset expression / value of the clip, relative to the layer.

By default, the OffsetY property is empty, which indicates the value of 0 ( top side of the layer ). The OffsetY property is 0, if the expression is missing or invalid.

For instance:

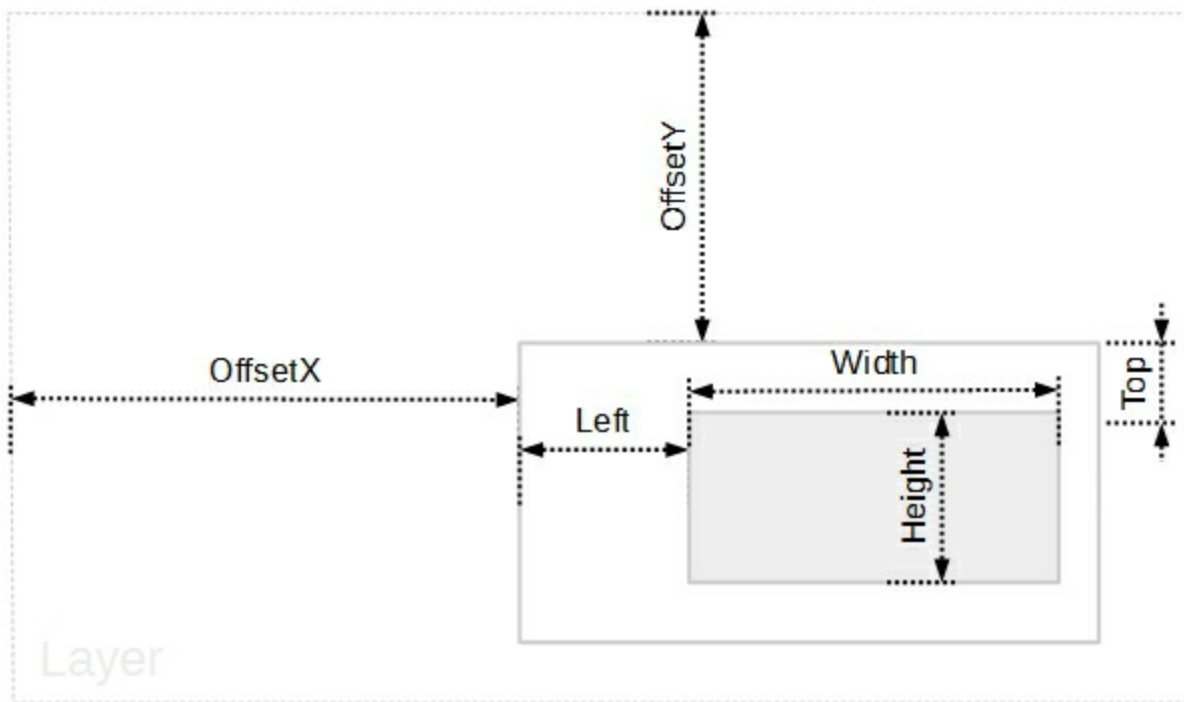
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRectangle.Top as String

Specifies the top position / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the top position / expression of the clip, relative to the layer.

By default, the Top property is empty, which indicates the value of 0 ( top side of the layer ). The Top property is 0, if the expression is missing or invalid.

For instance:

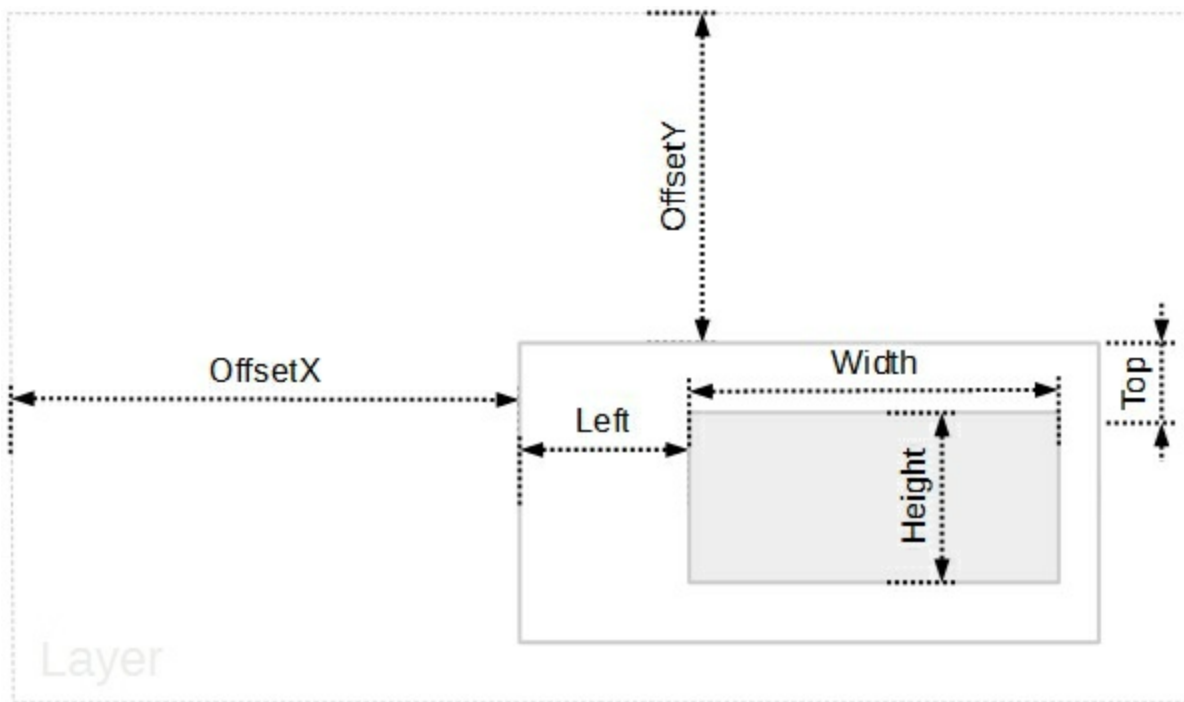
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipRectangle.Width as String

Specifies the width value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that specifies the width value / expression of the clip, relative to the layer.

By default, the Width property is empty, which indicates the width of the layer . The Width property is "width", if the expression is missing or invalid.

For instance:

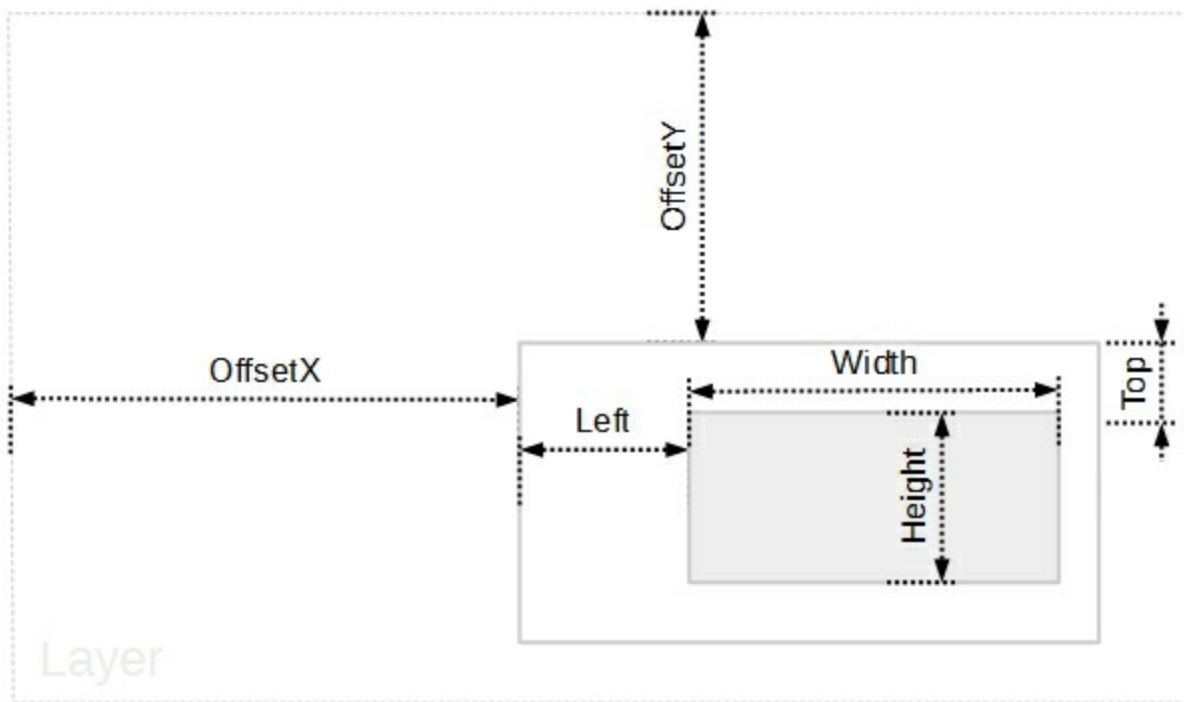
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

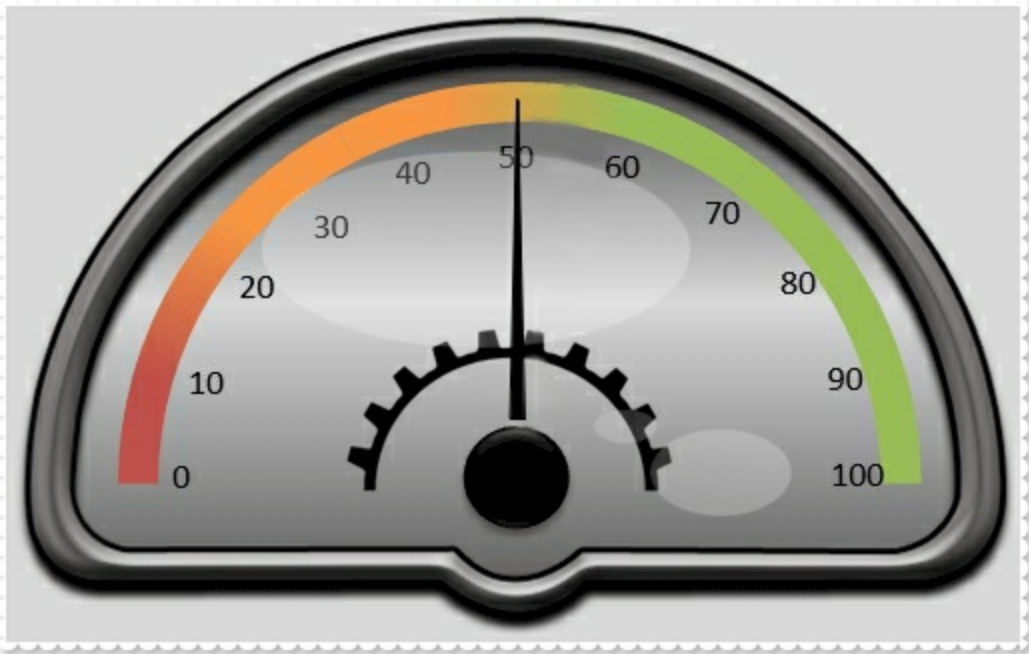
If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

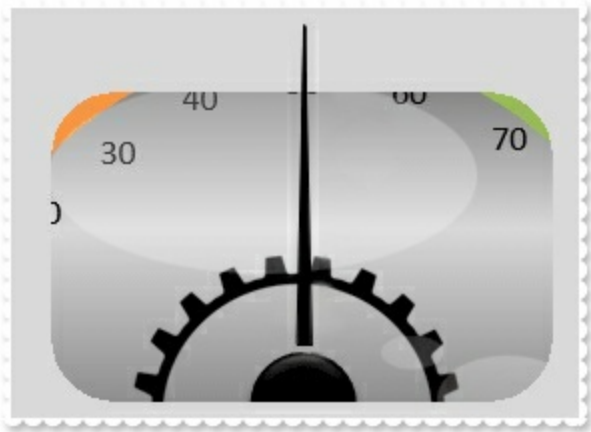
# ClipRoundRectangle object

The ClipRoundRectangle object holds information about an rectangular clip region.

For instance, having the following gauge:



a round rectangular clip region over the background layer shows as:



The ClipRoundRectangle property supports the following properties and methods:

Name	Description
<a href="#">Height</a>	Specifies the height value / expression of the clip, relative to the layer.
<a href="#">InverseClip</a>	Indicates whether the current clip object is inverted.
<a href="#">Left</a>	Specifies the left position / expression of the clip, relative to the layer.
<a href="#">OffsetX</a>	Specifies the x-offset expression / value of the clip, relative to the layer.

[OffsetY](#)

Specifies the y-offset expression / value of the clip, relative to the layer.

[RoundRadiusX](#)

Specifies the x-radius value / expression of the round corner, relative to the layer.

[RoundRadiusY](#)

Specifies the y-radius value / expression of the round corner, relative to the layer.

[Top](#)

Specifies the top position / expression of the clip, relative to the layer.

[Width](#)

Specifies the width value / expression of the clip, relative to the layer.

# property ClipRoundRectangle.Height as String

Specifies the height value / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the height value / expression of the clip, relative to the layer.

By default, the Height property is empty, which indicates the height of the layer. The Height property is "height", if the expression is missing or invalid.

For instance:

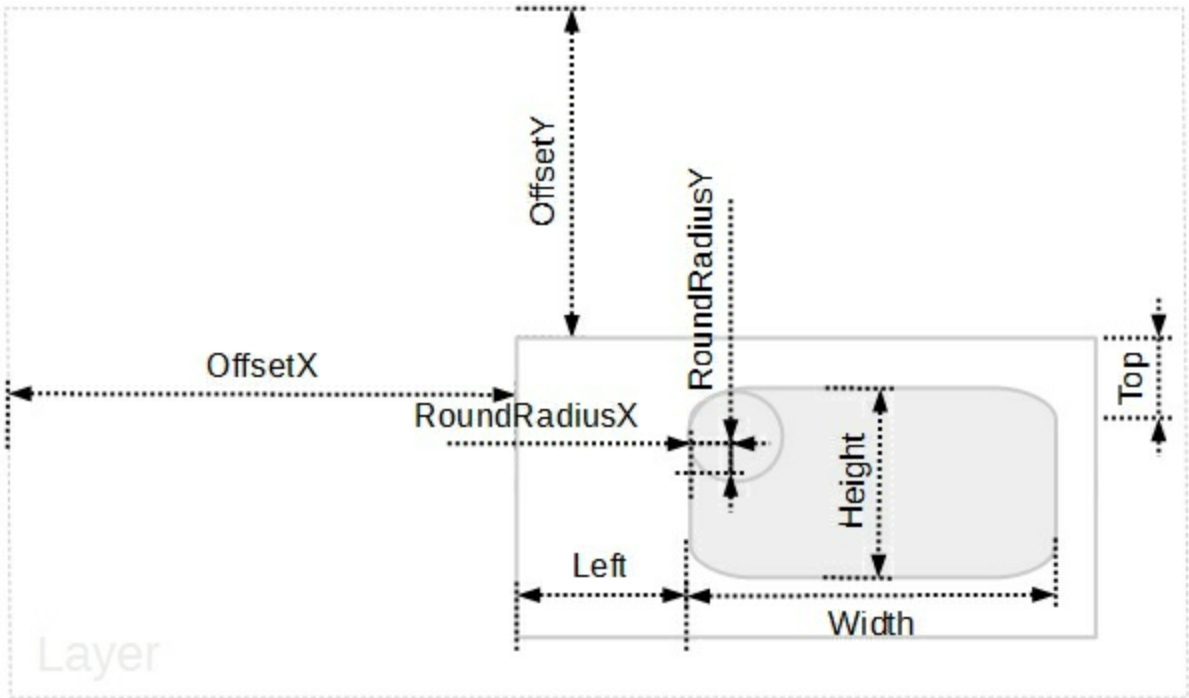
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- Height, Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRoundRectangle.InverseClip as Boolean

Indicates whether the current clip object is inverted.

Type	Description
Boolean	A Boolean expression that indicates whether the current clip object is inverted.

By default, InverseClip property is False. The InverseClip property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

To define or specify the position / size of the rectangular clip object you can use any of the following properties:

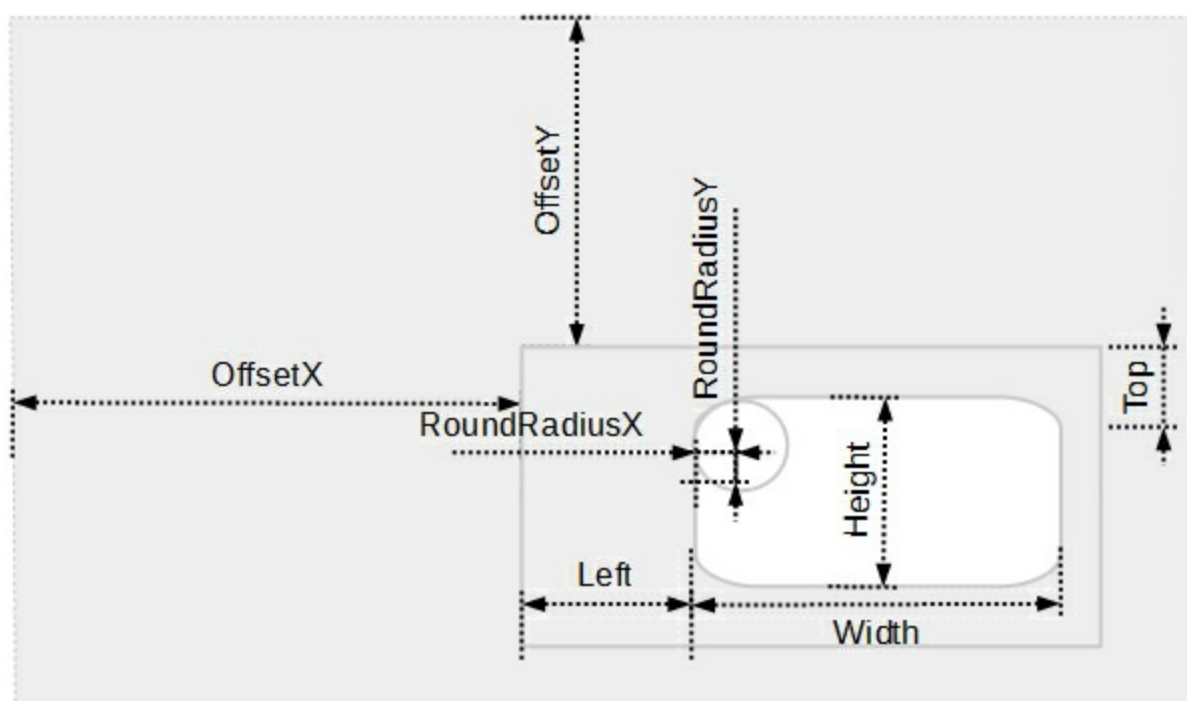
- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The following screen shot shows properties of the clipping objects relative to the layer:





# property ClipRoundRectangle.Left as String

Specifies the left position / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the left position / expression of the clip, relative to the layer.

By default, the Left property is empty, which indicates the value of 0 ( left side of the layer ). The Left property is 0, if the expression is missing or invalid.

For instance:

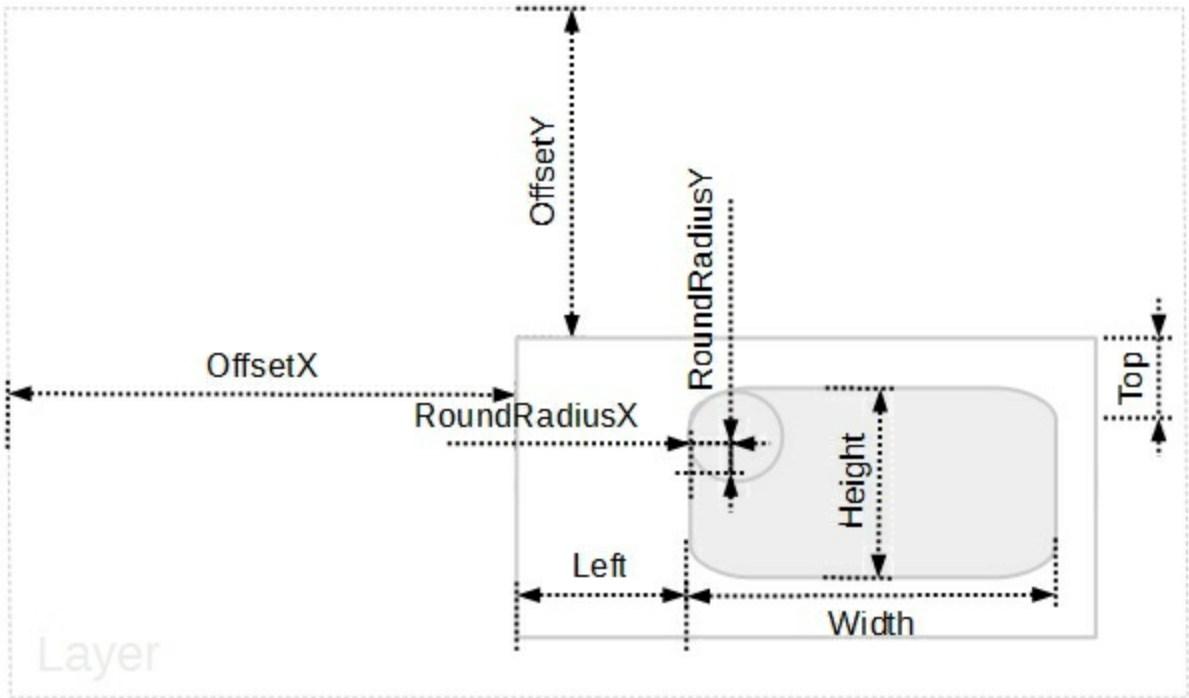
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRoundRectangle.OffsetX as String

Specifies the x-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the x-offset expression / value of the clip, relative to the layer.

By default, the OffsetX property is empty, which indicates the value of 0 ( left side of the layer ). The OffsetX property is 0, if the expression is missing or invalid.

For instance:

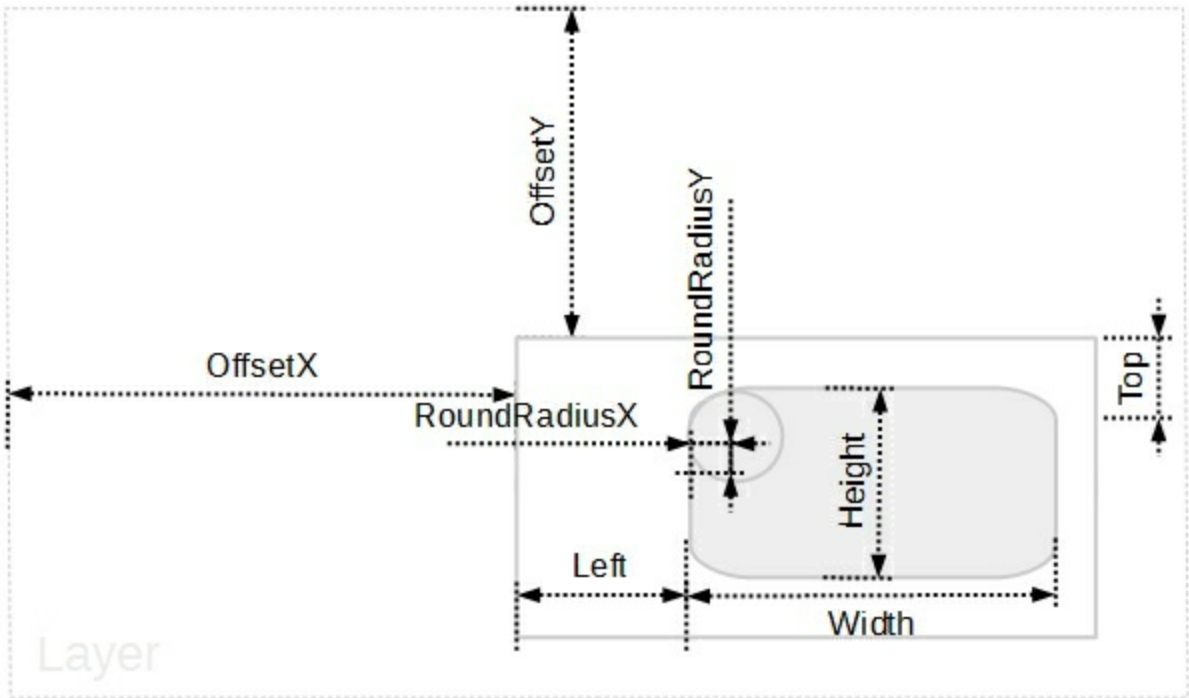
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRoundRectangle.OffsetY as String

Specifies the y-offset expression / value of the clip, relative to the layer.

Type	Description
String	A String expression that defines the y-offset expression / value of the clip, relative to the layer.

By default, the OffsetY property is empty, which indicates the value of 0 ( top side of the layer ). The OffsetY property is 0, if the expression is missing or invalid.

For instance:

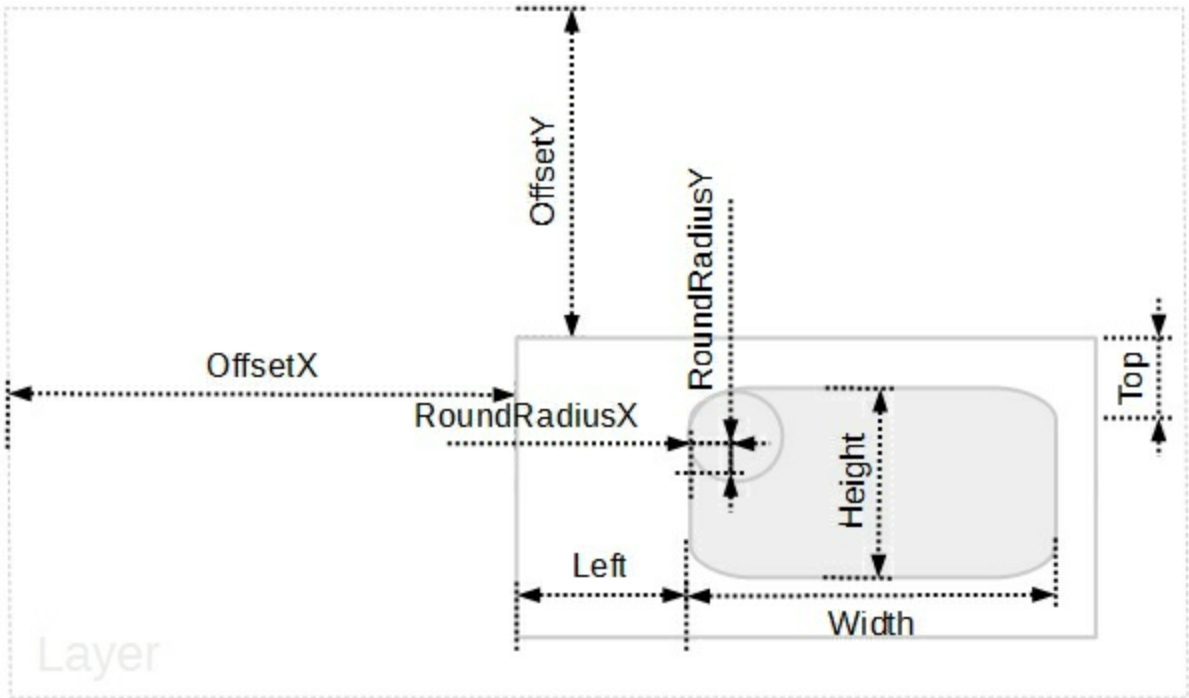
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRoundRectangle.RoundRadiusX as String

Specifies the x-radius value / expression of the round corner, relative to the layer.

Type	Description
String	A String expression that defines the x-radius value / expression of the round corner, relative to the layer.

By default, the RoundRadiusX property is empty, which indicates 0 ( no round corner ). The Round RadiusX property is 0 (no round corner ), if the expression is empty, missing or invalid.

For instance:

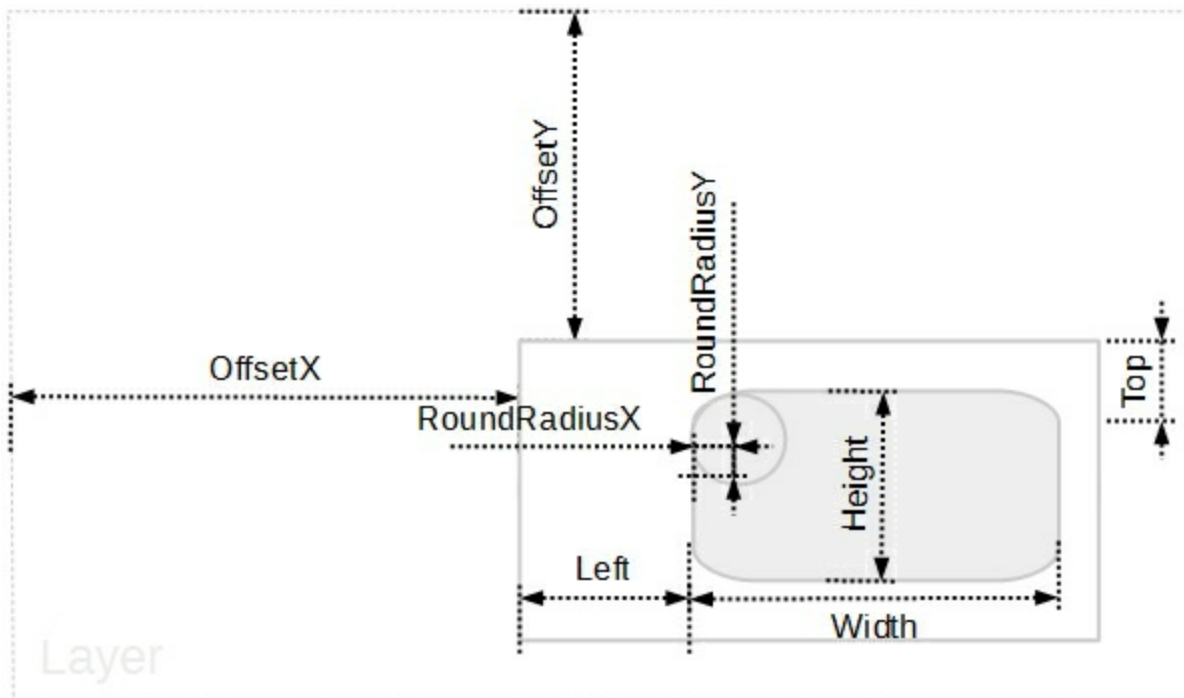
- "15", 15 pixels radius
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.



# property ClipRoundRectangle.RoundRadiusY as String

Specifies the y-radius value / expression of the round corner, relative to the layer.

Type	Description
String	A String expression that defines the y-radius value / expression of the round corner, relative to the layer.

By default, the RoundRadiusY property is empty, which indicates 0 ( no round corner ). The RoundRadiusY property is 0 ( no round corner ), if the expression is empty, missing or invalid.

For instance:

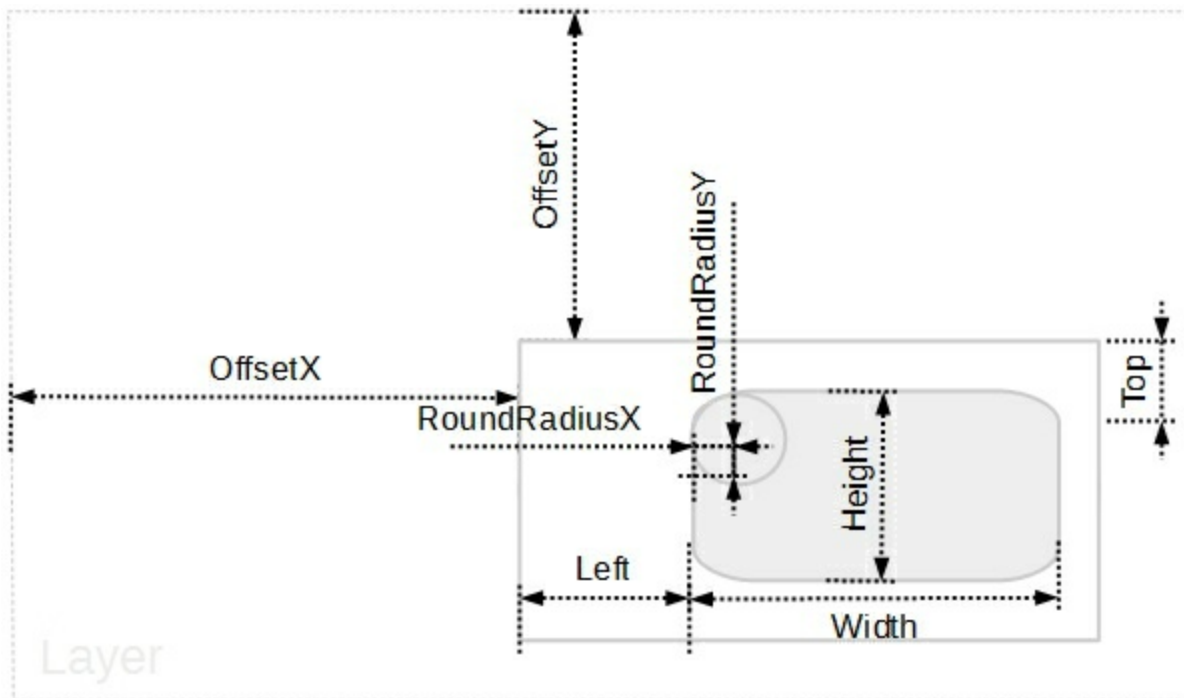
- "15", 15 pixels radius
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or right side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRoundRectangle.Top as String

Specifies the top position / expression of the clip, relative to the layer.

Type	Description
String	A String value that specifies the top position / expression of the clip, relative to the layer.

By default, the Top property is empty, which indicates the value of 0 ( top side of the layer ). The Top property is 0, if the expression is missing or invalid.

For instance:

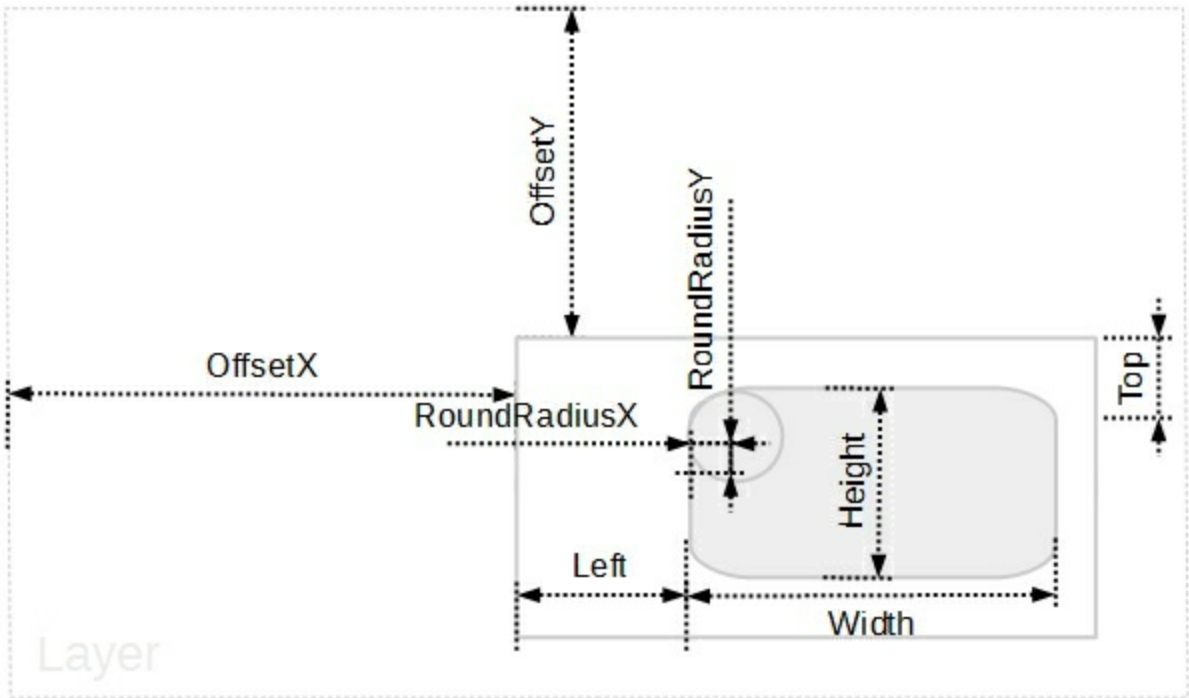
- "-15", 15 pixels up to the top side of the layer
- "height / 2" indicates the half of the layer's height or the y-center of the layer
- "height", indicates the height of the layer or bottom side of the layer
- "value / 100 \* height", indicates the [Value](#) percent of height of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* height" expression gets a quarter of the height of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

# property ClipRoundRectangle.Width as String

Specifies the width value / expression of the clip, relative to the layer.

Type	Description
String	A String expression that specifies the width value / expression of the clip, relative to the layer.

By default, the Width property is empty, which indicates the width of the layer . The Width property is "width", if the expression is missing or invalid.

For instance:

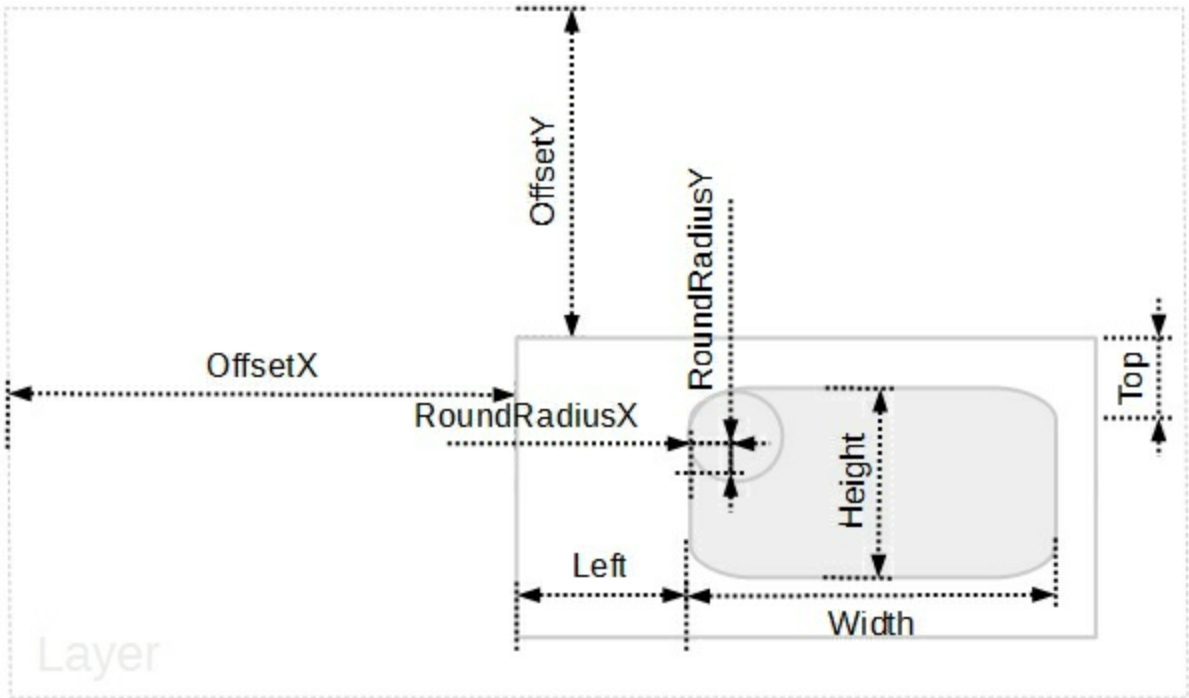
- "-15", 15 pixels to the left side of the layer
- "width / 2" indicates the half of the layer's width or the x-center of the layer
- "width", indicates the width of the layer or right side of the layer
- "value / 100 \* width", indicates the [Value](#) percent of width of the layer, so if clip's [Value](#) percent is 25, the "value / 100 \* width" expression gets a quarter of the width of the layer, or 50 gets half of it, and so on.

This property supports the following keywords:

- **value** keyword specifies the clip's value pointed by the clip's [Value](#) property
- **width** or **lwidth** keywords, indicates the width in pixels of the layer
- **height** or **lheight** keywords, indicates the height in pixels of the layer

Also, this property supports all constants, operators and functions defined [here](#).

The following screen shot shows properties of the clipping objects relative to the layer:



To define or specify the position / size of the rectangular clip object you can use any of the following properties:

- [Left](#), Specifies the left position / expression of the clip, relative to the layer.
- [Top](#), Specifies the top position / expression of the clip, relative to the layer.
- [Width](#), Specifies the width value / expression of the clip, relative to the layer.
- [Height](#), Specifies the height value / expression of the clip, relative to the layer.
- [RoundRadiusX](#), Specifies the x-radius value / expression of the round corner, relative to the layer.
- [RoundRadiusY](#), Specifies the y-radius value / expression of the round corner, relative to the layer.

To move the clipping region you can use any of the following properties:

- [OffsetX](#), Specifies the x-offset expression / value of the clip, relative to the layer.
- [OffsetY](#), Specifies the y-offset expression / value of the clip, relative to the layer.

If none of these properties are calling no clipping is applied to layer.

The [InverseClip](#) property inverses the current clipping region, so anything that was included in the clipping region will be excluded, and reverse.

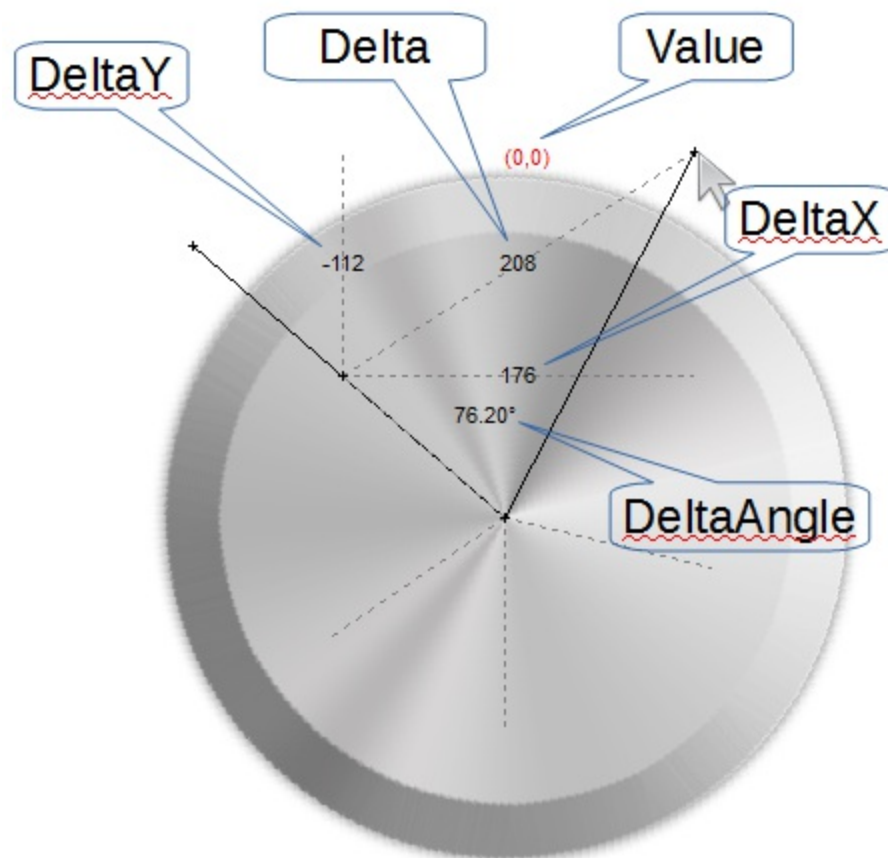
# DragInfo object

The DragInfo object holds information about dragging operation. Currently, the DragInfo object can be accessed through the drag events. Any layer on the control supports drag operations like moving, rotation, or combination of them, when the user clicks and drags a layer. The drag operation automatically starts when the user clicks a visible, selectable and draggable layer. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). The [Visible](#) property shows or hides a specific layer (visible). The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [Change](#) event occurs when the layer's value is changed.

The control fires the drag events in the following order:

- [DragStart](#) event, notifies that a layer begins to drag. You can use the DragStart event to cancel the dragging operation.
- [Drag](#) event, notifies that the layer is dragging. You can use the Drag event to perform other actions, on any layer during the dragging operation.
- [DragEnd](#) event notifies, the dragging the layer ends. You can use the DragEnd event to perform other actions, on any layer when dragging operation ends.

The following screen shot shows a few information ( angle, offset, values, ... ) you can get during dragging operation:



The DragInfo object supports the following properties and methods:

Name	Description
<a href="#">Button</a>	Specifies the button that initiated the drag operation.
<a href="#">Clockwise</a>	Indicates if the rotation is clockwise or anticlockwise.
<a href="#">CumulativeRotateAngle</a>	Indicates the cumulative rotation angle.
<a href="#">CurrentX</a>	Indicates the current x-position of the cursor, while dragging the layer.
<a href="#">CurrentY</a>	Indicates the current y-position of the cursor, while dragging the layer.
<a href="#">Debug</a>	Specifies debugging information to be shown while dragging the layers.
<a href="#">Delta</a>	Returns the distance between clicking and current points.
<a href="#">DeltaAngle</a>	Returns the rotation angle.
<a href="#">DeltaX</a>	Returns the offset on the x-coordinate of the the current drag operation.
<a href="#">DeltaY</a>	Returns the offset on the y-coordinate of the the current drag operation.
<a href="#">Layer</a>	Specifies the layer being dragged.
<a href="#">RotateAngleValid</a>	Validates the rotation angle of the layer, during dragging.
<a href="#">UserData</a>	Indicates any extra data associated with the dragging data.
<a href="#">X</a>	Indicates the x-position of the cursor, when the drag operation starts.
<a href="#">Y</a>	Indicates the y-position of the cursor, when the drag operation starts.



# property DragInfo.Button as Long

Specifies the button that initiated the drag operation.

Type	Description
Long	A Long expression that specifies the button that initiated the drag operation. 1 indicates the left mouse button, while 2 indicates the right mouse button.

The Button property indicates the button that initiated the drag operation. The Button property is read-only. The drag operation can start if clicking with left or right mouse button any visible / selectable / draggable layer in the control. For instance, you can disable dragging with the right mouse button by changing the Change parameter of the [DragStart](#) event, to True, if the Button property is 2. The drag operation ends when the user releases the mouse button, or the user presses the ESC key. The [DragEnd](#) event notifies that the dragging the layer ends.

# property DragInfo.Clockwise as Boolean

Indicates if the rotation is clockwise or anticlockwise.

Type	Description
Boolean	A Boolean expression that specifies whether the rotation is clockwise or anticlockwise.

The Clockwise property indicates if the rotation is clockwise or anticlockwise. A clockwise (typically abbreviated as CW) motion is one that proceeds in the same direction as a clock's hands: from the top to the right, then down and then to the left, and back up to the top. The Clockwise property is updated once you start rotating the object. The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation. The [CumulativeRotateAngle](#) property specifies the cumulative rotation angle, during the dragging operation. The [RotateAngle](#) property specifies the current angle of the rotation of the specified layer. The [RotateAngleValid](#) property specifies an expression that validates the rotation angle of the layer, during dragging operation.

# property DragInfo.CumulativeRotateAngle as Double

Indicates the cumulative rotation angle.

Type	Description
Double	A Double expression that specifies the cumulative rotation angle.

The CumulativeRotateAngle property specifies the cumulative rotation angle, during the dragging operation. The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation. The [RotateAngleValid](#) property specifies an expression that validates the rotation angle of the layer, during dragging operation. The [RotateAngle](#) property specifies the current angle of the rotation of the specified layer. The [Clockwise](#) property indicates if the rotation is clockwise or anticlockwise. A clockwise (typically abbreviated as CW) motion is one that proceeds in the same direction as a clock's hands: from the top to the right, then down and then to the left, and back up to the top.

# property DragInfo.CurrentX as Long

Indicates the current x-position of the cursor, while dragging the layer.

Type	Description
Long	A Long expression that indicates the current x-position of the cursor, while dragging the layer.

The CurrentX / [CurrentY](#) property indicates the current (x,y)-position of the cursor, relative to the upper-left corner of the control, while dragging the layer. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the OffsetYValid property on "0", and so no vertical movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- CurrentX property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

# property DragInfo.CurrentY as Long

Indicates the current y-position of the cursor, while dragging the layer.

Type	Description
Long	A Long expression that indicates the current y-position of the cursor, while dragging the layer.

The [CurrentX](#) / CurrentY property indicates the current (x,y)-position of the cursor, relative to the upper-left corner of the control, while dragging the layer. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the OffsetXValid property on "0", and so no horizontal movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- CurrentY property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

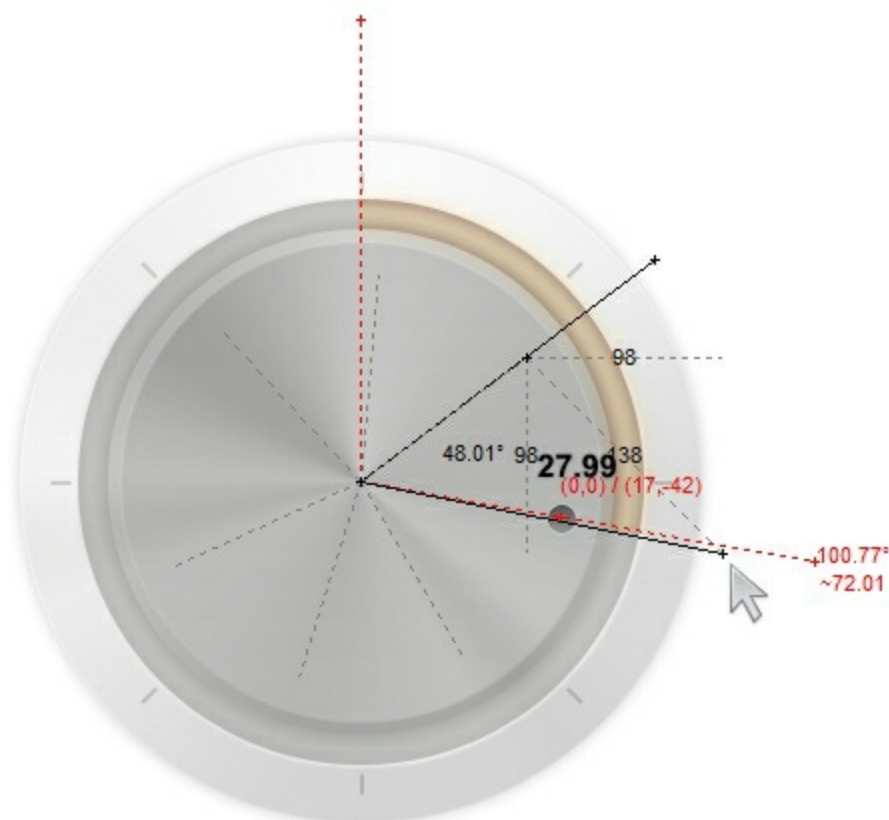
## property DragInfo.Debug as DebugLayerDragEnum

Specifies debugging information to be shown while dragging the layers.

Type	Description
<a href="#">DebugLayerDragEnum</a>	A DebugLayerDragEnum expression that specifies the information to be included when debugging the drag operation.

By default, the Debug property is `exDebugLayerDragNothing`, so no debug information is displayed during the drag operation. The Debug property specifies debugging information to be shown while dragging the layers. The Debug property should be called during the [DragStart](#) event, or whenever debugging information should be displayed. The debugging information includes offsets, angles, values, and so on. Use the [Debug](#) property of the Control to display layers in debug mode. During drag operation you can use the [RotateAngleValid](#) property to limit the rotation angle.

The following information shows all debug information while dragging the layer:



## property DragInfo.Delta as Double

Returns the distance between clicking and current points.

Type	Description
Double	A Double expression that specifies the distance between (CurrentX,CurrentY) and (X,Y) points.

The Delta property, returns the distance between clicking and current points. The [CurrentX](#) / [CurrentY](#) property indicates the current (x,y)-position of the cursor, relative to the upper-left corner of the control, while dragging the layer. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the OffsetYValid property on "0", and so no vertical movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- Delta property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

## property DragInfo.DeltaAngle as Double

Returns the rotation angle.

Type	Description
Double	A double expression that specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

The DeltaAngle property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation. The Clockwise property indicates if the rotation is clockwise or anticlockwise. A clockwise (typically abbreviated as CW) motion is one that proceeds in the same direction as a clock's hands: from the top to the right, then down and then to the left, and back up to the top. The [RotateAngle](#) property specifies the current angle of the rotation of the specified layer. The [CumulativeRotateAngle](#) property specifies the cumulative rotation angle, during the dragging operation. The [RotateAngleValid](#) property specifies an expression that validates the rotation angle of the layer, during dragging operation. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the OffsetYValid property on "0", and so no vertical movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.



## property DragInfo.DeltaX as Long

Returns the offset on the x-coordinate of the the current drag operation.

Type	Description
Long	A Long expression that returns the offset on the x-coordinate of the the current drag operation.

The DeltaX / [DeltaY](#) property indicates the current (horizontal, vertical)-offset of the movement during dragging operation. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the OffsetYValid property on "0", and so no vertical movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- DeltaX property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

# property DragInfo.DeltaY as Long

Returns the offset on the y-coordinate of the the current drag operation.

Type	Description
Long	A Long expression that returns the offset on the y-coordinate of the the current drag operation.

The [DeltaX](#) / DeltaY property indicates the current (horizontal, vertical)-offset of the movement during dragging operation. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the OffsetXValid property on "0", and so no horizontal movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of CurrentX - X.
- DeltaY property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of CurrentY - Y.

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

# property DragInfo.Layer as Long

Specifies the layer being dragged.

Type	Description
Long	A Long expression that specifies the index of the layer being dragged.

By default, the Layer property indicates the layer being clicked when the drag operation begins. You can change the Layer property to perform the drag operation to any other layer. In order to do that, you need to change the layer's OnDrag property. The [OnDrag](#) property indicates the action to be performed when the user drags the layer. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor.

The following samples show how you can rotate the layer with the index 9, by clicking anywhere on the control:

## VBA (MS Access, Excell...)

```
' DragStart event - Occurs once the user starts dragging a layer.
Private Sub Gauge1_DragStart(ByVal DragInfo As Object,Cancel As Boolean)
    ' DragInfo.Layer = 9
    ' Layers(DragInfo.Layer).OnDrag = 2
End Sub

With Gauge1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
End With
```

## VB6

```
' DragStart event - Occurs once the user starts dragging a layer.
Private Sub Gauge1_DragStart(ByVal DragInfo As EXGAUGELibCtl.IDragInfo,Cancel As
Boolean)
    ' DragInfo.Layer = 9
    ' Layers(DragInfo.Layer).OnDrag = 2
End Sub
```

With Gauge1

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

.PicturesName = "`Layer` + int(value + 1) + `.png`"

.Layers.Count = 11

End With

## VB.NET

*' DragStart event - Occurs once the user starts dragging a layer.*

Private Sub Exgauge1\_DragStart(ByVal sender As System.Object, ByVal DragInfo As  
exontrol.EXGAUGELib.DragInfo, ByRef Cancel As Boolean) Handles

Exgauge1.DragStart

*' DragInfo.Layer = 9*

*' Layers(DragInfo.Layer).OnDrag = 2*

End Sub

With Exgauge1

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

.PicturesName = "`Layer` + int(value + 1) + `.png`"

.Layers.Count = 11

End With

## VB.NET for /COM

*' DragStart event - Occurs once the user starts dragging a layer.*

Private Sub AxGauge1\_DragStart(ByVal sender As System.Object, ByVal e As  
AxEXGAUGELib.\_IGaugeEvents\_DragStartEvent) Handles AxGauge1.DragStart

*' DragInfo.Layer = 9*

*' Layers(DragInfo.Layer).OnDrag = 2*

End Sub

With AxGauge1

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

.PicturesName = "`Layer` + int(value + 1) + `.png`"

.Layers.Count = 11

**C++**

```

// DragStart event - Occurs once the user starts dragging a layer.
void OnDragStartGauge1(LPDISPATCH DragInfo, BOOL FAR* Cancel)
{
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
}

/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
    Library'

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
>GetControlUnknown();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(11);

```

**C++ Builder**

```

// DragStart event - Occurs once the user starts dragging a layer.
void __fastcall TForm1::Gauge1DragStart(TObject *Sender, Exgaugelib_tlb::IDragInfo
*DragInfo, VARIANT_BOOL * Cancel)
{
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
}

Gauge1->PicturesPath = L"C:\\Program

```

```
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png";
Gauge1->Layers->Count = 11;
```

## C#

```
// DragStart event - Occurs once the user starts dragging a layer.
private void exgauge1_DragStart(object sender,exontrol.EXGAUGELib.DragInfo
DragInfo,ref bool Cancel)
{
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
}
//this.exgauge1.DragStart += new
exontrol.EXGAUGELib.exg2antt.DragStartEventHandler(this.exgauge1_DragStart);

exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
exgauge1.PicturesName = "Layer` + int(value + 1) + `.png";
exgauge1.Layers.Count = 11;
```

## JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="DragStart(DragInfo,Cancel)"
LANGUAGE="JScript">
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
```

```

Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1.PicturesName = "\\Layer` + int(value + 1) + `.png`;
Gauge1.Layers.Count = 11;
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_DragStart(DragInfo,Cancel)
    ' DragInfo.Layer = 9
    ' Layers(DragInfo.Layer).OnDrag = 2
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob"
        .PicturesName = "\\Layer` + int(value + 1) + `.png`"
        .Layers.Count = 11
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```
// DragStart event - Occurs once the user starts dragging a layer.
```

```

private void axGauge1_DragStart(object sender,
AxEXGAUGELib._IGaugeEvents_DragStartEvent e)
{
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
}
//this.axGauge1.DragStart += new
AxEXGAUGELib._IGaugeEvents_DragStartEventHandler(this.axGauge1_DragStart);

axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "\\Layer` + int(value + 1) + `.png`;
axGauge1.Layers.Count = 11;

```

## X++ (Dynamics Ax 2009)

```

// DragStart event - Occurs once the user starts dragging a layer.
void onEvent_DragStart(COM _DragInfo,COMVariant /*bool*/ _Cancel)
{
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
    ;
}

public void init()
{
    ;

    super();

    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("\\Layer` + int(value + 1) + `.png");
    exgauge1.Layers().Count(11);
}

```

## Delphi 8 (.NET only)



*// DragStart event - Occurs once the user starts dragging a layer.*

```
procedure TWinForm1.AxGauge1_DragStart(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_DragStartEvent);
begin
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
end;

with AxGauge1 do
begin
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := `Layer` + int(value + 1) + `.png`;
    Layers.Count := 11;
end
```

## Delphi (standard)

*// DragStart event - Occurs once the user starts dragging a layer.*

```
procedure TForm1.Gauge1DragStart(ASender: TObject; DragInfo : IDragInfo;var
Cancel : WordBool);
begin
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2
end;

with Gauge1 do
begin
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := `Layer` + int(value + 1) + `.png`;
    Layers.Count := 11;
end
```

## VFP

*\*\*\* DragStart event - Occurs once the user starts dragging a layer. \*\*\**

LPARAMETERS DragInfo,Cancel

```

*** DragInfo.Layer = 9
*** Layers(DragInfo.Layer).OnDrag = 2

```

```

with thisform.Gauge1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
endwith

```

## dBASE Plus

```

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    DragStart = class::nativeObject_DragStart
endwith
*/
// Occurs once the user starts dragging a layer.
function nativeObject_DragStart(DragInfo,Cancel)
    /* DragInfo.Layer = 9 */
    /* Layers(DragInfo.Layer).OnDrag = 2 */
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    return

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11

```

## XBasic (Alpha Five)

```

' Occurs once the user starts dragging a layer.
function DragStart as v (DragInfo as OLE::Exontrol.Gauge.1::IDragInfo,Cancel as L)
    ' DragInfo.Layer = 9

```

```

' Layers(DragInfo.Layer).OnDrag = 2
oGauge = topparent:CONTROL_ACTIVEX1.activex
end function

Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11

```

## Visual Objects

```

METHOD OCX_Exontrol1DragStart(DragInfo,Cancel) CLASS MainDialog
    // DragStart event - Occurs once the user starts dragging a layer.
    // DragInfo.Layer = 9
    // Layers(DragInfo.Layer).OnDrag = 2

RETURN NIL

oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 11

```

## PowerBuilder

```

/*begin event DragStart(oleobject DragInfo,boolean Cancel) - Occurs once the user
starts dragging a layer.*/
/*
    DragInfo.Layer = 9
    Layers(DragInfo.Layer).OnDrag = 2
    oGauge = ole_1.Object
*/

```

```
/*end event DragStart*/
```

OleObject oGauge

oGauge = ole\_1.Object

oGauge.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

oGauge.PicturesName = ""Layer` + int(value + 1) + `.png`"

oGauge.Layers.Count = 11

## Visual DataFlex

```
// Occurs once the user starts dragging a layer.
```

```
Procedure OnComDragStart Variant IIDragInfo Boolean IICancel
```

```
Forward Send OnComDragStart IIDragInfo IICancel
```

```
// DragInfo.Layer = 9
```

```
// Layers(DragInfo.Layer).OnDrag = 2
```

```
End_Procedure
```

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Set ComPicturesPath to "C:\Program
```

```
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
Set ComPicturesName to ""Layer` + int(value + 1) + `.png`"
```

```
Variant voLayers
```

```
Get ComLayers to voLayers
```

```
Handle hoLayers
```

```
Get Create (RefClass(cComLayers)) to hoLayers
```

```
Set pvComObject of hoLayers to voLayers
```

```
Set ComCount of hoLayers to 11
```

```
Send Destroy to hoLayers
```

```
End_Procedure
```

## XBase++

```
PROCEDURE OnDragStart(oGauge,DragInfo,Cancel)
```

```
/*DragInfo.Layer = 9*/
```

```
/*Layers(DragInfo.Layer).OnDrag = 2*/
```

```
RETURN
```

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oGauge
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oGauge := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-83ED1790AAD3}*/
```

```
    oGauge:create(,, {10,60},{610,370} )
```

```
    oGauge:DragStart := {|| DragInfo,Cancel| OnDragStart(oGauge,DragInfo,Cancel)}
```

```
/*Occurs once the user starts dragging a layer.*/
```

```
    oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
```

```
    oGauge:Layers():Count := 11
```

```
    oForm:Show()
```

```
    DO WHILE nEvent != xbeP_Quit
```

```
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
        oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
    ENDDO
```

```
RETURN
```



# property DragInfo.RotateAngleValid as String

Validates the rotation angle of the layer, during dragging.

Type	Description
String	A String that indicates the expression to validate the rotation angle of the layer, during dragging.

The RotateAngleValid property specifies an expression that validates the rotation angle of the layer, during dragging operation. If using, the RotateAngleValid property should be called during the [DragStart](#) event. For instance, it is known that any layer can be rotated from 0 to 360% degree, but what about limitation of the rotation based on the mouse movement. In other words, if you rotate the clockwise the layer you want to prevent the layer to exceed the max value, so the RotateAngleValid property does the trick. The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation. The [RotateAngle](#) property specifies the current angle of the rotation of the specified layer. The [CumulativeRotateAngle](#) property specifies the cumulative rotation angle, during the dragging operation. The [Clockwise](#) property indicates if the rotation is clockwise or anticlockwise. A clockwise (typically abbreviated as CW) motion is one that proceeds in the same direction as a clock's hands: from the top to the right, then down and then to the left, and back up to the top. The [RotateAngleValid](#) property validates / limits the rotation angle of the layer.

For instance, RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)", validates the rotation-angle so it always will be between 0 and 360, in other words limits the rotation angle.

*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.*

The **value** keyword indicates the cumulative rotation angle ( [CumulativeRotateAngle](#) property ), during the dragging operation.

*The constants are ( DPI-Aware components ):*

- **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 :*

- **MIN** ( 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.
- **MAX** ( 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 than *if* (immediate if operator) alternative.

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

**expression case ([default : default\_expression ; ] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)**

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

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

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance `type(%1) = 8` specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

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

- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"
- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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

How can I limit the rotation from 0 to 360 degree, while dragging?

**VBA (MS Access, Excell...)**

' *DragStart* event - Occurs once the user starts dragging a layer.

```

Private Sub Gauge1_DragStart(ByVal DragInfo As Object,Cancel As Boolean)
    ' DragInfo.Debug = 483
    ' DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
End Sub

With Gauge1
    With .Layers.Add("back")
        .RotateType = 2
        .Left = "(width-512)/2"
        .Top = "(height-512)/2"
        .Height = 512
        .Width = 512
        With .Background.Picture
            .Value = "c:\exontrol\images\card.png"
            .Left = "(width-pwidth)/2"
            .Top = "(height-pheight)/2"
            .Width = "pwidth"
            .Height = "pheight"
        End With
        .OnDrag = 2
        .RotateAngle = -45
    End With
End With

```

## VB6

```

' DragStart event - Occurs once the user starts dragging a layer.
Private Sub Gauge1_DragStart(ByVal DragInfo As EXGAUGELibCtl.IDragInfo,Cancel As Boolean)
    ' DragInfo.Debug = 483
    ' DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
End Sub

With Gauge1
    With .Layers.Add("back")
        .RotateType = exRotateBilinearInterpolation
        .Left = "(width-512)/2"
    End With
End With

```



```

.Top = "(height-512)/2"
.Height = 512
.Width = 512
With .Background.Picture
    .Value = "c:\exontrol\images\card.png"
    .Left = "(width-pwidth)/2"
    .Top = "(height-pheight)/2"
    .Width = "pwidth"
    .Height = "pheight"
End With
.OnDrag = exDoRotate
.RotateAngle = -45
End With
End With

```

## VB.NET

*' DragStart event - Occurs once the user starts dragging a layer.*

```

Private Sub Exgauge1_DragStart(ByVal sender As System.Object, ByVal DragInfo As
exontrol.EXGAUGELib.DragInfo, ByRef Cancel As Boolean) Handles
Exgauge1.DragStart

```

```


```

```

    ' DragInfo.Debug = 483

```

```

    ' DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"

```

```

End Sub

```

```

With Exgauge1

```

```

    With .Layers.Add("back")

```

```

        .RotateType =

```

```

exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation

```

```

        .Left = "(width-512)/2"

```

```

        .Top = "(height-512)/2"

```

```

        .Height = 512

```

```

        .Width = 512

```

```

        With .Background.Picture

```

```

            .Value = "c:\exontrol\images\card.png"

```

```

            .Left = "(width-pwidth)/2"

```

```

            .Top = "(height-pheight)/2"

```

```

        .Width = "pwidth"
        .Height = "pheight"
    End With
    .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
    .RotateAngle = -45
End With
End With

```

## VB.NET for /COM

```

' DragStart event - Occurs once the user starts dragging a layer.
Private Sub AxGauge1_DragStart(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_DragStartEvent) Handles AxGauge1.DragStart
    ' DragInfo.Debug = 483
    ' DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
End Sub

With AxGauge1
    With .Layers.Add("back")
        .RotateType = EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
        .Left = "(width-512)/2"
        .Top = "(height-512)/2"
        .Height = 512
        .Width = 512
        With .Background.Picture
            .Value = "c:\exontrol\images\card.png"
            .Left = "(width-pwidth)/2"
            .Top = "(height-pheight)/2"
            .Width = "pwidth"
            .Height = "pheight"
        End With
        .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
        .RotateAngle = -45
    End With
End With

```

*// DragStart event - Occurs once the user starts dragging a layer.*

```
void OnDragStartGauge1(LPDISPATCH DragInfo, BOOL FAR* Cancel)
```

```
{
```

```
    // DragInfo.Debug = 483
```

```
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
```

```
}
```

```
/*
```

*Copy and paste the following directives to your header file as it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control Library'*

```
#import <ExGauge.dll>
```

```
using namespace EXGAUGELib;
```

```
*/
```

```
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
```

```
> GetControlUnknown();
```

```
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->Add("back");
```

```
var_Layer->PutRotateType(EXGAUGELib::exRotateBilinearInterpolation);
```

```
var_Layer->PutLeft(L"(width-512)/2");
```

```
var_Layer->PutTop(L"(height-512)/2");
```

```
var_Layer->PutHeight(L"512");
```

```
var_Layer->PutWidth(L"512");
```

```
EXGAUGELib::ILPicturePtr var_Picture = var_Layer->GetBackground()->GetPicture();
```

```
var_Picture->PutValue("c:\\exontrol\\images\\card.png");
```

```
var_Picture->PutLeft(L"(width-pwidth)/2");
```

```
var_Picture->PutTop(L"(height-pheight)/2");
```

```
var_Picture->PutWidth(L"pwidth");
```

```
var_Picture->PutHeight(L"pheight");
```

```
var_Layer->PutOnDrag(EXGAUGELib::exDoRotate);
```

```
var_Layer->PutRotateAngle(-45);
```

**C++ Builder**

*// DragStart event - Occurs once the user starts dragging a layer.*

```

void __fastcall TForm1::Gauge1DragStart(TObject *Sender,Exgaugelib_tlb::IDragInfo
*DragInfo,VARIANT_BOOL * Cancel)
{
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
}

Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->Add(TVariant("back"));
var_Layer->RotateType =
Exgaugelib_tlb::RotateTypeEnum::exRotateBilinearInterpolation;
var_Layer->Left = L"(width-512)/2";
var_Layer->Top = L"(height-512)/2";
var_Layer->Height = L"512";
var_Layer->Width = L"512";
Exgaugelib_tlb::ILPicturePtr var_Picture = var_Layer->Background->Picture;
var_Picture->set_Value(TVariant("c:\\exontrol\\images\\card.png"));
var_Picture->Left = L"(width-pwidth)/2";
var_Picture->Top = L"(height-pheight)/2";
var_Picture->Width = L"pwidth";
var_Picture->Height = L"pheight";
var_Layer->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
var_Layer->RotateAngle = -45;

```

## C#

```

// DragStart event - Occurs once the user starts dragging a layer.
private void exgauge1_DragStart(object sender,exontrol.EXGAUGELib.DragInfo
DragInfo,ref bool Cancel)
{
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
}
//this.exgauge1.DragStart += new
exontrol.EXGAUGELib.exg2antt.DragStartEventHandler(this.exgauge1_DragStart);

```

```

exontrol.EXGAUGELib.Layer var_Layer = exgaug1.Layers.Add("back");
    var_Layer.RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
    var_Layer.Left = "(width-512)/2";
    var_Layer.Top = "(height-512)/2";
    var_Layer.Height = 512.ToString();
    var_Layer.Width = 512.ToString();
    exontrol.EXGAUGELib.Picture var_Picture = var_Layer.Background.Picture;
    var_Picture.Value = "c:\\exontrol\\images\\card.png";
    var_Picture.Left = "(width-pwidth)/2";
    var_Picture.Top = "(height-pheight)/2";
    var_Picture.Width = "pwidth";
    var_Picture.Height = "pheight";
    var_Layer.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer.RotateAngle = -45;

```

## JScript/JavaScript

```

<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="DragStart(DragInfo,Cancel)"
LANGUAGE="JScript">
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Layer = Gauge1.Layers.Add("back");
    var_Layer.RotateType = 2;
    var_Layer.Left = "(width-512)/2";

```

```

var_Layer.Top = "(height-512)/2";
var_Layer.Height = 512;
var_Layer.Width = 512;
var var_Picture = var_Layer.Background.Picture;
    var_Picture.Value = "c:\\exontrol\\images\\card.png";
    var_Picture.Left = "(width-pwidth)/2";
    var_Picture.Top = "(height-pheight)/2";
    var_Picture.Width = "pwidth";
    var_Picture.Height = "pheight";
var_Layer.OnDrag = 2;
var_Layer.RotateAngle = -45;
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_DragStart(DragInfo,Cancel)
    ' DragInfo.Debug = 483
    ' DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        With .Layers.Add("back")
            .RotateType = 2
            .Left = "(width-512)/2"
            .Top = "(height-512)/2"
            .Height = 512

```

```

.Width = 512
With .Background.Picture
.Value = "c:\exontrol\images\card.png"
.Left = "(width-pwidth)/2"
.Top = "(height-pheight)/2"
.Width = "pwidth"
.Height = "pheight"
End With
.OnDrag = 2
.RotateAngle = -45
End With
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

// DragStart event - Occurs once the user starts dragging a layer.
private void axGauge1_DragStart(object sender,
AxEXGAUGELib._IGaugeEvents_DragStartEvent e)
{
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
}
//this.axGauge1.DragStart += new
AxEXGAUGELib._IGaugeEvents_DragStartEventHandler(this.axGauge1_DragStart);

EXGAUGELib.Layer var_Layer = axGauge1.Layers.Add("back");
var_Layer.RotateType =
EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
var_Layer.Left = "(width-512)/2";
var_Layer.Top = "(height-512)/2";
var_Layer.Height = 512.ToString();
var_Layer.Width = 512.ToString();

```

```

EXGAUGELib.Picture var_Picture = var_Layer.Background.Picture;
var_Picture.Value = "c:\\exontrol\\images\\card.png";
var_Picture.Left = "(width-pwidth)/2";
var_Picture.Top = "(height-pheight)/2";
var_Picture.Width = "pwidth";
var_Picture.Height = "pheight";
var_Layer.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
var_Layer.RotateAngle = -45;

```

## X++ (Dynamics Ax 2009)

```

// DragStart event - Occurs once the user starts dragging a layer.
void onEvent_DragStart(COM _DragInfo,COMVariant /*bool*/ _Cancel)
{
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
    ;
}

public void init()
{
    COM com_Background,com_Layer,com_Picture;
    anytype var_Background,var_Layer,var_Picture;
    ;

    super();

    var_Layer = COM::createFromObject(exgauge1.Layers()).Add("back"); com_Layer =
var_Layer;
    com_Layer.RotateType(2/*exRotateBilinearInterpolation*/);
    com_Layer.Left("(width-512)/2");
    com_Layer.Top("(height-512)/2");
    com_Layer.Height(512);
    com_Layer.Width(512);
    var_Background = COM::createFromObject(com_Layer.Background());

```



```

com_Background = var_Background;
var_Picture = com_Background.Picture(); com_Picture = var_Picture;
com_Picture.Value("c:\\exontrol\\images\\card.png");
com_Picture.Left("(width-pwidth)/2");
com_Picture.Top("(height-pheight)/2");
com_Picture.Width("pwidth");
com_Picture.Height("pheight");
com_Layer.OnDrag(2/*exDoRotate*/);
com_Layer.RotateAngle(-45);
}

```

## Delphi 8 (.NET only)

```

// DragStart event - Occurs once the user starts dragging a layer.
procedure TForm1.AxGauge1_DragStart(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_DragStartEvent);
begin
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
end;

with AxGauge1 do
begin
    with Layers.Add('back') do
    begin
        RotateType := EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
        Left := '(width-512)/2';
        Top := '(height-512)/2';
        Height := 512;
        Width := 512;
        with Background.Picture do
        begin
            Value := 'c:\\exontrol\\images\\card.png';
            Left := '(width-pwidth)/2';
            Top := '(height-pheight)/2';
            Width := 'pwidth';

```

```

    Height := 'pheight';
end;
OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
RotateAngle := -45;
end;
end

```

## Delphi (standard)

```

// DragStart event - Occurs once the user starts dragging a layer.
procedure TForm1.Gauge1DragStart(ASender: TObject; DragInfo : IDragInfo;var
Cancel : WordBool);
begin
    // DragInfo.Debug = 483
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"
end;

with Gauge1 do
begin
    with Layers.Add('back') do
    begin
        RotateType := EXGAUGELib_TLB.exRotateBilinearInterpolation;
        Left := '(width-512)/2';
        Top := '(height-512)/2';
        Height := 512;
        Width := 512;
        with Background.Picture do
        begin
            Value := 'c:\exontrol\images\card.png';
            Left := '(width-pwidth)/2';
            Top := '(height-pheight)/2';
            Width := 'pwidth';
            Height := 'pheight';
        end;
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        RotateAngle := -45;
    end;
end;

```

```
end;  
end
```

## VFP

```
*** DragStart event - Occurs once the user starts dragging a layer. ***  
LPARAMETERS DragInfo,Cancel  
    *** DragInfo.Debug = 483  
    *** DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :  
value)"  
  
with thisform.Gauge1  
    with .Layers.Add("back")  
        .RotateType = 2  
        .Left = "(width-512)/2"  
        .Top = "(height-512)/2"  
        .Height = 512  
        .Width = 512  
        with .Background.Picture  
            .Value = "c:\exontrol\images\card.png"  
            .Left = "(width-pwidth)/2"  
            .Top = "(height-pheight)/2"  
            .Width = "pwidth"  
            .Height = "pheight"  
        endwhile  
        .OnDrag = 2  
        .RotateAngle = -45  
    endwhile  
endwith
```

## dBASE Plus

```
/*  
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)  
    DragStart = class::nativeObject_DragStart  
endwith  
*/  
// Occurs once the user starts dragging a layer.
```

```

function nativeObject_DragStart(DragInfo,Cancel)
    /* DragInfo.Debug = 483 */
    /* DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)" */
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
return

local oGauge,var_Layer,var_Picture

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
var_Layer = oGauge.Layers.Add("back")
    var_Layer.RotateType = 2
    var_Layer.Left = "(width-512)/2"
    var_Layer.Top = "(height-512)/2"
    var_Layer.Height = Str(512)
    var_Layer.Width = Str(512)
    var_Picture = var_Layer.Background.Picture
        var_Picture.Value = "c:\exontrol\images\card.png"
        var_Picture.Left = "(width-pwidth)/2"
        var_Picture.Top = "(height-pheight)/2"
        var_Picture.Width = "pwidth"
        var_Picture.Height = "pheight"
    var_Layer.OnDrag = 2
    var_Layer.RotateAngle = -45

```

## XBasic (Alpha Five)

```

' Occurs once the user starts dragging a layer.
function DragStart as v (DragInfo as OLE::Exontrol.Gauge.1::IDragInfo,Cancel as L)
    ' DragInfo.Debug = 483
    ' DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
    oGauge = topparent:CONTROL_ACTIVEX1.activex
end function

Dim oGauge as P
Dim var_Layer as P

```

```
Dim var_Picture as P
```

```
oGauge = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Layer = oGauge.Layers.Add("back")
```

```
var_Layer.RotateType = 2
```

```
var_Layer.Left = "(width-512)/2"
```

```
var_Layer.Top = "(height-512)/2"
```

```
var_Layer.Height = 512
```

```
var_Layer.Width = 512
```

```
var_Picture = var_Layer.Background.Picture
```

```
var_Picture.Value = "c:\exontrol\images\card.png"
```

```
var_Picture.Left = "(width-pwidth)/2"
```

```
var_Picture.Top = "(height-pheight)/2"
```

```
var_Picture.Width = "pwidth"
```

```
var_Picture.Height = "pheight"
```

```
var_Layer.OnDrag = 2
```

```
var_Layer.RotateAngle = -45
```

## Visual Objects

```
METHOD OCX_Exontrol1DragStart(DragInfo,Cancel) CLASS MainDialog
```

```
// DragStart event - Occurs once the user starts dragging a layer.
```

```
// DragInfo.Debug = 483
```

```
// DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
```

```
RETURN NIL
```

```
local var_Picture as ILPicture
```

```
local var_Layer as ILayer
```

```
var_Layer := oDCOCX_Exontrol1:Layers:Add("back")
```

```
var_Layer.RotateType := exRotateBilinearInterpolation
```

```
var_Layer.Left := "(width-512)/2"
```

```
var_Layer.Top := "(height-512)/2"
```

```
var_Layer.Height := AsString(512)
```

```

var_Layer:Width := AsString(512)
var_Picture := var_Layer:Background:Picture
  var_Picture:Value := "c:\exontrol\images\card.png"
  var_Picture:Left := "(width-pwidth)/2"
  var_Picture:Top := "(height-pheight)/2"
  var_Picture:Width := "pwidth"
  var_Picture:Height := "pheight"
var_Layer:OnDrag := exDoRotate
var_Layer:RotateAngle := -45

```

## PowerBuilder

```

/*begin event DragStart(oleobject DragInfo,boolean Cancel) - Occurs once the user
starts dragging a layer.*/
/*
  DragInfo.Debug = 483
  DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 : value)"
  oGauge = ole_1.Object
*/
/*end event DragStart*/

```

```
OleObject oGauge,var_Layer,var_Picture
```

```

oGauge = ole_1.Object
var_Layer = oGauge.Layers.Add("back")
  var_Layer.RotateType = 2
  var_Layer.Left = "(width-512)/2"
  var_Layer.Top = "(height-512)/2"
  var_Layer.Height = String(512)
  var_Layer.Width = String(512)
  var_Picture = var_Layer.Background.Picture
    var_Picture.Value = "c:\exontrol\images\card.png"
    var_Picture.Left = "(width-pwidth)/2"
    var_Picture.Top = "(height-pheight)/2"
    var_Picture.Width = "pwidth"
    var_Picture.Height = "pheight"

```

```
var_Layer.OnDrag = 2  
var_Layer.RotateAngle = -45
```

## Visual DataFlex

```
// Occurs once the user starts dragging a layer.  
Procedure OnComDragStart Variant IIDragInfo Boolean IICancel  
    Forward Send OnComDragStart IIDragInfo IICancel  
    // DragInfo.Debug = 483  
    // DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :  
value)"  
End_Procedure  
  
Procedure OnCreate  
    Forward Send OnCreate  
    Variant voLayers  
    Get ComLayers to voLayers  
    Handle hoLayers  
    Get Create (RefClass(cComLayers)) to hoLayers  
    Set pvComObject of hoLayers to voLayers  
    Variant voLayer  
    Get ComAdd of hoLayers "back" to voLayer  
    Handle hoLayer  
    Get Create (RefClass(cComLayer)) to hoLayer  
    Set pvComObject of hoLayer to voLayer  
        Set ComRotateType of hoLayer to OLEexRotateBilinearInterpolation  
        Set ComLeft of hoLayer to "(width-512)/2"  
        Set ComTop of hoLayer to "(height-512)/2"  
        Set ComHeight of hoLayer to 512  
        Set ComWidth of hoLayer to 512  
    Variant voBackground  
    Get ComBackground of hoLayer to voBackground  
    Handle hoBackground  
    Get Create (RefClass(cComBackground)) to hoBackground  
    Set pvComObject of hoBackground to voBackground  
    Variant voPicture
```

```

Get ComPicture of hoBackground to voPicture
Handle hoPicture
Get Create (RefClass(cComPicture)) to hoPicture
Set pvComObject of hoPicture to voPicture
    Set ComValue of hoPicture to "c:\exontrol\images\card.png"
    Set ComLeft of hoPicture to "(width-pwidth)/2"
    Set ComTop of hoPicture to "(height-pheight)/2"
    Set ComWidth of hoPicture to "pwidth"
    Set ComHeight of hoPicture to "pheight"
Send Destroy to hoPicture
Send Destroy to hoBackground
Set ComOnDrag of hoLayer to OLEexDoRotate
Set ComRotateAngle of hoLayer to -45
Send Destroy to hoLayer
Send Destroy to hoLayers
End_Procedure

```

## XBase++

```

PROCEDURE OnDragStart(oGauge,DragInfo,Cancel)
    /*DragInfo.Debug = 483*/
    /*DragInfo.RotateAngleValid = "value < 0 ? 0 : (value > 360 ? 359.999999 :
value)"/

RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge
    LOCAL oPicture
    LOCAL oLayer

    oForm := XbpDialog():new( AppDesktop() )

```



```
oForm:drawingArea:clipChildren := .T.  
oForm:create( ,, {100,100}, {640,480},,, .F. )  
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oGauge := XbpActiveXControl():new( oForm:drawingArea )  
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-  
83ED1790AAD3}*/  
oGauge:create(,, {10,60},{610,370} )
```

```
oGauge:DragStart := {|| DragInfo,Cancel| OnDragStart(oGauge,DragInfo,Cancel)}  
/*Occurs once the user starts dragging a layer.*/
```

```
oLayer := oGauge:Layers():Add("back")  
oLayer:RotateType := 2/*exRotateBilinearInterpolation*/  
oLayer:Left := "(width-512)/2"  
oLayer:Top := "(height-512)/2"  
oLayer:Height := Transform(512,"")  
oLayer:Width := Transform(512,"")  
oPicture := oLayer:Background():Picture()  
oPicture:Value := "c:\exontrol\images\card.png"  
oPicture:Left := "(width-pwidth)/2"  
oPicture:Top := "(height-pheight)/2"  
oPicture:Width := "pwidth"  
oPicture:Height := "pheight"  
oLayer:OnDrag := 2/*exDoRotate*/  
oLayer:RotateAngle := -45
```

```
oForm:Show()  
DO WHILE nEvent != xbeP_Quit  
    nEvent := AppEvent( @mp1, @mp2, @oXbp )  
    oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

# property DragInfo.UserData as Variant

Indicates any extra data associated with the dragging data.

Type	Description
Variant	A Variant expression that specifies any extra data associated with the drag operation.

By default, the UserData property is empty. Use the UserData property to store any extra data with the drag operation. You can set the UserData property during the [DragStart](#) event, and use it later during the [Drag](#) or [DragEnd](#) event. The [UserData](#) property of the Layer indicates any extra data associated with the layer.

# property DragInfo.X as Long

Indicates the x-position of the cursor, when the drag operation starts.

Type	Description
Long	A Long expression that indicates the x-position of the cursor, when the drag operation starts.

The [X](#) / [Y](#) property indicates the (x,y)-position of the cursor, relative to the upper-left corner of the control, when dragging operation begins. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the [OffsetYValid](#) property on "0", and so no vertical movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of  $\text{CurrentX} - X$ .
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of  $\text{CurrentY} - Y$ .

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

# property DragInfo.Y as Long

Indicates the y-position of the cursor, when the drag operation starts.

Type	Description
Long	A Long expression that indicates the y-position of the cursor, when the drag operation starts.

The [X](#) / [Y](#) property indicates the (x,y)-position of the cursor, relative to the upper-left corner of the control, when dragging operation begins. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. For instance, you can use the [OffsetXValid](#) property on "0", and so no horizontal movement is allowed.

The following properties can be used during dragging to determine the horizontal / vertical offset:

- [X](#) property indicates the x-position of the cursor, when the drag operation starts.
- [Y](#) property indicates the y-position of the cursor, when the drag operation starts.
- [CurrentX](#) property indicates the current x-position of the cursor, while dragging the layer.
- [CurrentY](#) property indicates the current y-position of the cursor, while dragging the layer.
- [Delta](#) property, returns the distance between clicking and current points.
- [DeltaX](#) property returns the offset on the x-coordinate of the the current drag operation, equivalent with the value of  $\text{CurrentX} - X$ .
- [DeltaY](#) property returns the offset on the y-coordinate of the the current drag operation, equivalent with the value of  $\text{CurrentY} - Y$ .

The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation.

# Foreground object

The Foreground object holds HTML captions to be shown on the layer's foreground. The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The layer's foreground can be visible or selectable. If not selectable, the user can not select it runtime, such as LayerFromPoint property ignores it. The Layer's foreground can display unlimited HTML captions of different sizes and positions.

The following screen shot shows all layer's background with a semi-transparent color, to highlight the layer's foreground:



The Foreground object supports the following properties and methods:

Name	Description
<a href="#">Caption</a>	Specifies the caption on the layer.
<a href="#">Color</a>	Specifies the layer's foreground color.
<a href="#">ExtraCaption</a>	Specifies any extra caption on the layer.
<a href="#">Selectable</a>	Returns or sets a value that indicates whether all objects on the layer's foreground are selectable.
<a href="#">Visible</a>	Specifies if the objects of the layer's foreground are shown or hidden.

# Property Foreground.Caption(Property as PropertyLayerCaptionEnum) as Variant

Specifies the caption on the layer.

Type	Description
Property as <a href="#">PropertyLayerCaptionEnum</a>	A PropertyLayerCaptionEnum expression that specifies the caption's property to be changed.
Variant	A VARIANT expression that specifies the value of the caption's property.

The control support unlimited HTML captions to be place anywhere on the control or on any layer of the control. The Caption( exLayerCaption) specifies the HTML caption to be shown on the control/layer. The [Images](#) method specifies the list of icons the control can display. The [HTMLPicture](#) adds or replaces a picture in HTML captions. The Caption(exLayerCaptionBackgroundExt) property indicates unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the control / layer's background. The caption on the control stay on its position, no matter what layer is moved or rotated, while a caption on a layer gets moved or rotated together with the layer itself. The [Color](#) property specifies the caption's foreground color.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows an extra-caption associated with the layer:



The following samples show how you can associate an extra-caption with a layer:

### VBA (MS Access, Excell...)

With Gauge1

.BeginUpdate

.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

.PicturesName = ""Layer` + int(value + 1) + `.png"

.Layers.Count = 10

With .Layers.Item(9)

.RotateType = 2

.OnDrag = 2

With .Foreground

.ExtraCaption("logo",3) = 2

.ExtraCaption("logo",8) = True

.ExtraCaption("logo",6) = "164"

.ExtraCaption("logo",4) = "width - 176"

.ExtraCaption("logo",5) = "-64"

.ExtraCaption("logo",0) = "<sha ;;0> <c> This is our logo<br> <c>

<img>logo</img> "

End With

```
End With
.EndUpdate
End With
```

## VB6

```
With Gauge1
.BeginUpdate
.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 10
With .Layers.Item(9)
.RotateType = exRotateBilinearInterpolation
.OnDrag = exDoRotate
With .Foreground
.ExtraCaption("logo",exLayerCaptionAnchor) = 2
.ExtraCaption("logo",exLayerCaptionWordWrap) = True
.ExtraCaption("logo",exLayerCaptionWidth) = "164"
.ExtraCaption("logo",exLayerCaptionLeft) = "width - 176"
.ExtraCaption("logo",exLayerCaptionTop) = "-64"
.ExtraCaption("logo",exLayerCaption) = "<sha ;;0> <c>This is our logo<br>
<c> <img>logo</img>"
End With
End With
.EndUpdate
End With
```

## VB.NET

```
With Exgauge1
.BeginUpdate()
.set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 10
```



```

With .Layers.Item(9)
    .RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
    .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
With .Foreground

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
- 176")

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
<sha ;;0> <c> This is our logo<br> <c> <img> logo</img>")
    End With
End With
    .EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 10

```

```

With .Layers.Item(9)
    .RotateType = EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
    .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
With .Foreground

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnchor)
= 2

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWordWr
= True

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidth)
= "164"

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft) =
"width - 176"

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop) =
"-64"
    .ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption)
= "<sha ;;0> <c> This is our logo<br> <c> <img>logo</img>"
    End With
End With
    .EndUpdate()
End With

```

## C++

```

/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
    Library'

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-

```

```

> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1-
> PutHTMLPicture(L"logo", "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(10);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(long(9));
    var_Layer->PutRotateType(EXGAUGELib::exRotateBilinearInterpolation);
    var_Layer->PutOnDrag(EXGAUGELib::exDoRotate);
    EXGAUGELib::IForegroundPtr var_Foreground = var_Layer->GetForeground();
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionAnchor, long(2));
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWordWrap, VARIANT_TRUE);
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWidth, "164");
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionLeft, "width - 176");
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionTop, "-64");
    var_Foreground->PutExtraCaption("logo", EXGAUGELib::exLayerCaption, "<sha
;;0> <c> This is our logo<br> <c> <img> logo </img>");
spGauge1->EndUpdate();

```

## C++ Builder

```

Gauge1->BeginUpdate();
Gauge1->HTMLPicture[L"logo"] =
TVariant("E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`;
Gauge1->Layers->Count = 10;
Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(9));

```

```

var_Layer->RotateType =
Exgaugelib_tlb::RotateTypeEnum::exRotateBilinearInterpolation;
var_Layer->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
Exgaugelib_tlb::IForegroundPtr var_Foreground = var_Layer->Foreground;
var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye
- 176"));
var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye
<sha ;;0> <c> This is our logo<br> <c> <img>logo</img>");
Gauge1->EndUpdate();

```

**C#**

```

exgauge1.BeginUpdate();
exgauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
exgauge1.Layers.Count = 10;
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[9];
var_Layer.RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
var_Layer.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;

```

```

exontrol.EXGAUGELib.Foreground var_Foreground = var_Layer.Foreground;

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

- 176");

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE
<sha ;;0> <c>This is our logo<br> <c> <img>logo</img>");
exgauge1.EndUpdate();

```

## JScript/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.HTMLPicture("logo") = "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png";
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    Gauge1.Layers.Count = 10;

```

```

var var_Layer = Gauge1.Layers.Item(9);
var_Layer.RotateType = 2;
var_Layer.OnDrag = 2;
var var_Foreground = var_Layer.Foreground;
var_Foreground.ExtraCaption("logo",3) = 2;
var_Foreground.ExtraCaption("logo",8) = true;
var_Foreground.ExtraCaption("logo",6) = "164";
var_Foreground.ExtraCaption("logo",4) = "width - 176";
var_Foreground.ExtraCaption("logo",5) = "-64";
var_Foreground.ExtraCaption("logo",0) = "<sha ;;0> <c>This is our logo<br>
<c> <img>logo</img>";
Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = ""Layer` + int(value + 1) + `.png""
        .Layers.Count = 10
        With .Layers.Item(9)
            .RotateType = 2
            .OnDrag = 2
            With .Foreground
                .ExtraCaption("logo",3) = 2

```

```

        .ExtraCaption("logo",8) = True
        .ExtraCaption("logo",6) = "164"
        .ExtraCaption("logo",4) = "width - 176"
        .ExtraCaption("logo",5) = "-64"
        .ExtraCaption("logo",0) = "<sha ;;0> <c> This is our logo<br> <c>
<img>logo</img>"
    End With
End With
.EndUpdate
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 10;
EXGAUGELib.Layer var_Layer = axGauge1.Layers[9];
    var_Layer.RotateType =
EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
    var_Layer.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    EXGAUGELib.Foreground var_Foreground = var_Layer.Foreground;

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

```

```

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa
- 176");

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa
<sha ;;0> <c>This is our logo<br> <c> <img>logo</img>");
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Foreground,com_Layer;
    anytype var_Foreground,var_Layer;
    ;

    super();

    exgauge1.BeginUpdate();
    exgauge1.HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(10);
    var_Layer =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(9));
com_Layer = var_Layer;
    com_Layer.RotateType(2/*exRotateBilinearInterpolation*/);
    com_Layer.OnDrag(2/*exDoRotate*/);
    var_Foreground = com_Layer.Foreground(); com_Foreground = var_Foreground;

    com_Foreground.ExtraCaption("logo",3/*exLayerCaptionAnchor*/,COMVariant::createF

```



```

com_Foreground.ExtraCaption("logo",8/*exLayerCaptionWordWrap*/,COMVariant::crea

    com_Foreground.ExtraCaption("logo",6/*exLayerCaptionWidth*/,"164");
    com_Foreground.ExtraCaption("logo",4/*exLayerCaptionLeft*/,"width - 176");
    com_Foreground.ExtraCaption("logo",5/*exLayerCaptionTop*/,"-64");
    com_Foreground.ExtraCaption("logo",0/*exLayerCaption*/,"<sha ;;0> <c> This
is our logo<br> <c> <img> logo</img>");
    exGauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
    BeginUpdate();
    set_HTMLPicture('logo','E:\Exontrol\Exontrol.Logo\exontrol.logo.png');
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 10;
    with Layers.Item[TObject(9)] do
    begin
        RotateType := EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
        OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
        with Foreground do
        begin

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnchor]
:= TObject(2);

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWordWra
:= TObject(True);

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidth]
:= '164';

```

```

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft] :=
'width - 176';

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop] :=
'-64';

    ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption]
:= ' <sha ;;0> <c>This is our logo<br> <c> <img>logo</img>';
    end;
end;
EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
    BeginUpdate();
    HTMLPicture['logo'] := 'E:\Exontrol\Exontrol.Logo\exontrol.logo.png';
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 10;
    with Layers.Item[OleVariant(9)] do
    begin
        RotateType := EXGAUGELib_TLB.exRotateBilinearInterpolation;
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        with Foreground do
        begin
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionAnchor] := OleVariant(2);
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWordWrap] :=
OleVariant(True);
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWidth] := '164';
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionLeft] := 'width - 176';
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionTop] := '-64';
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaption] := ' <sha ;;0> <c>This is
our logo<br> <c> <img>logo</img>';
        end;
    end;
end;

```

```

end;
EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.Object.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 10
with .Layers.Item(9)
.RotateType = 2
.OnDrag = 2
with .Foreground
.ExtraCaption("logo",3) = 2
.ExtraCaption("logo",8) = .T.
.ExtraCaption("logo",6) = "164"
.ExtraCaption("logo",4) = "width - 176"
.ExtraCaption("logo",5) = "-64"
.ExtraCaption("logo",0) = "<sha ;;0> <c> This is our logo<br> <c>
<img> logo </img>"
endwith
endwith
.EndUpdate
endwith

```

## dBASE Plus

```

local oGauge,var_Foreground,var_Layer

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.Template = [HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"] // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"

```

```

oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Item(9)
var_Layer.RotateType = 2
var_Layer.OnDrag = 2
var_Foreground = var_Layer.Foreground
// var_Foreground.ExtraCaption("logo",3) = 2
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",3) = 2]
endwith
// var_Foreground.ExtraCaption("logo",8) = true
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",8) = True]
endwith
// var_Foreground.ExtraCaption("logo",6) = "164"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",6) = "164"]
endwith
// var_Foreground.ExtraCaption("logo",4) = "width - 176"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",4) = "width - 176"]
endwith
// var_Foreground.ExtraCaption("logo",5) = "-64"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",5) = "-64"]

```

```

endwith
// var_Foreground.ExtraCaption("logo",0) = "<sha ;;0><c>This is our logo<br>
<c><img>logo</img>"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",0) = "<sha ;;0><c>This is our
logo<br><c><img>logo</img>"]
endwith
oGauge.EndUpdate()

```

## XBasic (Alpha Five)

```

Dim oGauge as P
Dim var_Foreground as P
Dim var_Layer as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.Template = "HTMLPicture(`logo`) =
'E:\Exontrol\Exontrol.Logo\exontrol.logo.png" // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Item(9)
    var_Layer.RotateType = 2
    var_Layer.OnDrag = 2
    var_Foreground = var_Layer.Foreground
        ' var_Foreground.ExtraCaption("logo",3) = 2
    oGauge.TemplateDef = "dim var_Foreground"
    oGauge.TemplateDef = var_Foreground
    oGauge.Template = "var_Foreground.ExtraCaption(`logo`,3) = 2"

    ' var_Foreground.ExtraCaption("logo",8) = .t.

```

```

oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,8) = True"

' var_Foreground.ExtraCaption("logo",6) = "164"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,6) = `164`"

' var_Foreground.ExtraCaption("logo",4) = "width - 176"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,4) = `width - 176`"

' var_Foreground.ExtraCaption("logo",5) = "-64"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,5) = `-64`"

' var_Foreground.ExtraCaption("logo",0) = "<sha ;;0> <c>This is our logo<br>
<c><img>logo</img>"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,0) = `<sha ;;0> <c>This
is our logo<br> <c><img>logo</img>`"

oGauge.EndUpdate()

```

## Visual Objects

```

local var_Foreground as IForeground
local var_Layer as ILayer

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:[HTMLPicture,"logo"] :=
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"

```

```

oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 10
var_Layer := oDCOCX_Exontrol1:Layers:[Item,9]
    var_Layer:RotateType := exRotateBilinearInterpolation
    var_Layer:OnDrag := exDoRotate
    var_Foreground := var_Layer:Foreground
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionAnchor] := 2
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionWordWrap] := true
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionWidth] := "164"
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionLeft] := "width - 176"
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionTop] := "-64"
        var_Foreground:[ExtraCaption,"logo",exLayerCaption] := "<sha ;;0> <c>This is
our logo<br> <c> <img>logo</img>"
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge,var_Foreground,var_Layer

oGauge = ole_1.Object
oGauge.BeginUpdate()
oGauge.HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Item(9)
    var_Layer.RotateType = 2
    var_Layer.OnDrag = 2
    var_Foreground = var_Layer.Foreground
        var_Foreground.ExtraCaption("logo",3,2)
        var_Foreground.ExtraCaption("logo",8,true)
        var_Foreground.ExtraCaption("logo",6,"164")
        var_Foreground.ExtraCaption("logo",4,"width - 176")

```

```
var_Foreground.ExtraCaption("logo",5,"-64")
var_Foreground.ExtraCaption("logo",0,"<sha ;;0> <c>This is our logo<br> <c>
<img>logo</img>")
oGauge.EndUpdate()
```

## Visual DataFlex

### Procedure OnCreate

```
Forward Send OnCreate
Send ComBeginUpdate
Set ComHTMLPicture "logo" to "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers
Get ComLayers to voLayers
Handle hoLayers
Get Create (RefClass(cComLayers)) to hoLayers
Set pvComObject of hoLayers to voLayers
Set ComCount of hoLayers to 10
Send Destroy to hoLayers
Variant voLayers1
Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
Variant voLayer
Get ComItem of hoLayers1 9 to voLayer
Handle hoLayer
Get Create (RefClass(cComLayer)) to hoLayer
Set pvComObject of hoLayer to voLayer
Set ComRotateType of hoLayer to OLEexRotateBilinearInterpolation
Set ComOnDrag of hoLayer to OLEexDoRotate
Variant voForeground
Get ComForeground of hoLayer to voForeground
Handle hoForeground
```



```

Get Create (RefClass(cComForeground)) to hoForeground
Set pvComObject of hoForeground to voForeground
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionAnchor to
2
    Set ComExtraCaption of hoForeground "logo"
OLEexLayerCaptionWordWrap to True
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionWidth to
"164"
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionLeft to
"width - 176"
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionTop to
"-64"
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaption to "<sha
;;0> <c>This is our logo<br> <c> <img>logo</img> "
    Send Destroy to hoForeground
    Send Destroy to hoLayer
    Send Destroy to hoLayers1
    Send ComEndUpdate
End_Procedure

```

## XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge
    LOCAL oForeground
    LOCAL oLayer

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„ .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

```

```

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()

oGauge:SetProperty("HTMLPicture","logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.p

    oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 10
    oLayer := oGauge:Layers:Item(9)
        oLayer:RotateType := 2/*exRotateBilinearInterpolation*/
        oLayer:OnDrag := 2/*exDoRotate*/
        oForeground := oLayer:Foreground()

oForeground:SetProperty("ExtraCaption","logo",3/*exLayerCaptionAnchor*/,2)

oForeground:SetProperty("ExtraCaption","logo",8/*exLayerCaptionWordWrap*/,.T.)

oForeground:SetProperty("ExtraCaption","logo",6/*exLayerCaptionWidth*/,"164")

oForeground:SetProperty("ExtraCaption","logo",4/*exLayerCaptionLeft*/,"width -
176")

oForeground:SetProperty("ExtraCaption","logo",5/*exLayerCaptionTop*/,"-64")
    oForeground:SetProperty("ExtraCaption","logo",0/*exLayerCaption*/,"<sha
;;0> <c> This is our logo<br> <c> <img> logo</img>")
    oGauge:EndUpdate()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO

```



# Property Foreground.Color as Color

Specifies the layer's foreground color.

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

By default, the Color property is -1, which indicates that has no effect. The Color property specifies the caption's foreground color. The [Visible](#) property shows or hides all layer's caption / extra-captions. The [Selectable](#) property makes all layer' captions / extra-captions selectable or unselectable.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

# property Foreground.ExtraCaption(Key as Variant, Property as PropertyLayerCaptionEnum) as Variant

Specifies any extra caption on the layer.

Type	Description
Key as Variant	Any VARIANT expression that indicates the key of the extra caption. You can use any value to identify your extra caption.
Property as <a href="#">PropertyLayerCaptionEnum</a>	A PropertyLayerCaptionEnum expression that specifies the extra caption's property to be changed.
Variant	A VARIANT expression that specifies the value of the extra caption's property.

The control support unlimited HTML captions to be place anywhere on the control or on any layer of the control. The Caption( exLayerCaption) specifies the HTML caption to be shown on the control/layer. The [Images](#) method specifies the list of icons the control can display. The [HTMLPicture](#) adds or replaces a picture in HTML captions. The Caption(exLayerCaptionBackgroundExt) property indicates unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the control / layer's background. The caption on the control stay on its position, no matter what layer is moved or rotated, while a caption on a layer gets moved or rotated together with the layer itself.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows an extra-caption associated with the layer:



The following samples show how you can associate an extra-caption with a layer:

### VBA (MS Access, Excell...)

With Gauge1

.BeginUpdate

.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

.PicturesName = ""Layer` + int(value + 1) + `.png"

.Layers.Count = 10

With .Layers.Item(9)

.RotateType = 2

.OnDrag = 2

With .Foreground

.ExtraCaption("logo",3) = 2

.ExtraCaption("logo",8) = True

.ExtraCaption("logo",6) = "164"

.ExtraCaption("logo",4) = "width - 176"

.ExtraCaption("logo",5) = "-64"

.ExtraCaption("logo",0) = "<sha ;;0> <c> This is our logo<br> <c>

<img>logo</img> "

End With

```
End With
.EndUpdate
End With
```

## VB6

```
With Gauge1
.BeginUpdate
.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 10
With .Layers.Item(9)
.RotateType = exRotateBilinearInterpolation
.OnDrag = exDoRotate
With .Foreground
.ExtraCaption("logo",exLayerCaptionAnchor) = 2
.ExtraCaption("logo",exLayerCaptionWordWrap) = True
.ExtraCaption("logo",exLayerCaptionWidth) = "164"
.ExtraCaption("logo",exLayerCaptionLeft) = "width - 176"
.ExtraCaption("logo",exLayerCaptionTop) = "-64"
.ExtraCaption("logo",exLayerCaption) = "<sha ;;0> <c>This is our logo<br>
<c> <img>logo</img>"
End With
End With
.EndUpdate
End With
```

## VB.NET

```
With Exgauge1
.BeginUpdate()
.set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 10
```

```

With .Layers.Item(9)
    .RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
    .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
With .Foreground

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
- 176")

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
<sha ;;0> <c> This is our logo<br> <c> <img> logo</img>")
    End With
End With
    .EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 10

```



```

With .Layers.Item(9)
    .RotateType = EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
    .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
With .Foreground

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnchor)
= 2

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWordWr
= True

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidth)
= "164"

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft) =
"width - 176"

.ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop) =
"-64"
    .ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption)
= "<sha ;;0> <c> This is our logo<br> <c> <img>logo</img>"
    End With
End With
.EndUpdate()
End With

```

## C++

```

/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
    Library'

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-

```

```

> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1-
> PutHTMLPicture(L"logo", "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(10);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(long(9));
    var_Layer->PutRotateType(EXGAUGELib::exRotateBilinearInterpolation);
    var_Layer->PutOnDrag(EXGAUGELib::exDoRotate);
    EXGAUGELib::IForegroundPtr var_Foreground = var_Layer->GetForeground();
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionAnchor, long(2));
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWordWrap, VARIANT_TRUE);
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWidth, "164");
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionLeft, "width - 176");
    var_Foreground-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionTop, "-64");
    var_Foreground->PutExtraCaption("logo", EXGAUGELib::exLayerCaption, "<sha
;;0> <c> This is our logo<br> <c> <img> logo</img>");
spGauge1->EndUpdate();

```

## C++ Builder

```

Gauge1->BeginUpdate();
Gauge1->HTMLPicture[L"logo"] =
TVariant("E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`;
Gauge1->Layers->Count = 10;
Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(9));

```

```

var_Layer->RotateType =
Exgaugelib_tlb::RotateTypeEnum::exRotateBilinearInterpolation;
var_Layer->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
Exgaugelib_tlb::IForegroundPtr var_Foreground = var_Layer->Foreground;
var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye
- 176"));
var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye

var_Foreground-
> set_ExtraCaption(TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLaye
<sha ;;0> <c> This is our logo<br> <c> <img>logo</img>");
Gauge1->EndUpdate();

```

## C#

```

exgauge1.BeginUpdate();
exgauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
exgauge1.Layers.Count = 10;
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[9];
var_Layer.RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
var_Layer.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;

```

```

exontrol.EXGAUGELib.Foreground var_Foreground = var_Layer.Foreground;

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

- 176");

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE

var_Foreground.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionE
<sha ;;0> <c>This is our logo<br> <c> <img>logo</img>");
exgauge1.EndUpdate();

```

## JScript/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.HTMLPicture("logo") = "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png";
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    Gauge1.Layers.Count = 10;

```

```

var var_Layer = Gauge1.Layers.Item(9);
var_Layer.RotateType = 2;
var_Layer.OnDrag = 2;
var var_Foreground = var_Layer.Foreground;
var_Foreground.ExtraCaption("logo",3) = 2;
var_Foreground.ExtraCaption("logo",8) = true;
var_Foreground.ExtraCaption("logo",6) = "164";
var_Foreground.ExtraCaption("logo",4) = "width - 176";
var_Foreground.ExtraCaption("logo",5) = "-64";
var_Foreground.ExtraCaption("logo",0) = "<sha ;;0> <c>This is our logo<br>
<c> <img>logo</img>";
Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = ""Layer` + int(value + 1) + `.png""
        .Layers.Count = 10
        With .Layers.Item(9)
            .RotateType = 2
            .OnDrag = 2
            With .Foreground
                .ExtraCaption("logo",3) = 2

```

```

        .ExtraCaption("logo",8) = True
        .ExtraCaption("logo",6) = "164"
        .ExtraCaption("logo",4) = "width - 176"
        .ExtraCaption("logo",5) = "-64"
        .ExtraCaption("logo",0) = "<sha ;;0> <c> This is our logo<br> <c>
<img>logo</img>"
    End With
End With
.EndUpdate
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 10;
EXGAUGELib.Layer var_Layer = axGauge1.Layers[9];
    var_Layer.RotateType =
EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
    var_Layer.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    EXGAUGELib.Foreground var_Foreground = var_Layer.Foreground;

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

```

```

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa
- 176");

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa

var_Foreground.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLa
<sha ;;0> <c>This is our logo<br> <c> <img>logo</img>");
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Foreground,com_Layer;
    anytype var_Foreground,var_Layer;
    ;

    super();

    exgauge1.BeginUpdate();
    exgauge1.HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(10);
    var_Layer =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(9));
com_Layer = var_Layer;
    com_Layer.RotateType(2/*exRotateBilinearInterpolation*/);
    com_Layer.OnDrag(2/*exDoRotate*/);
    var_Foreground = com_Layer.Foreground(); com_Foreground = var_Foreground;

    com_Foreground.ExtraCaption("logo",3/*exLayerCaptionAnchor*/,COMVariant::createF

```

```

com_Foreground.ExtraCaption("logo",8/*exLayerCaptionWordWrap*/,COMVariant::crea

    com_Foreground.ExtraCaption("logo",6/*exLayerCaptionWidth*/,"164");
    com_Foreground.ExtraCaption("logo",4/*exLayerCaptionLeft*/,"width - 176");
    com_Foreground.ExtraCaption("logo",5/*exLayerCaptionTop*/,"-64");
    com_Foreground.ExtraCaption("logo",0/*exLayerCaption*/,"<sha ;;0> <c> This
is our logo<br> <c> <img> logo</img>");
    exGauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
    BeginUpdate();
    set_HTMLPicture('logo','E:\Exontrol\Exontrol.Logo\exontrol.logo.png');
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 10;
    with Layers.Item[TObject(9)] do
    begin
        RotateType := EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
        OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
        with Foreground do
        begin

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnchor]
:= TObject(2);

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWordWra
:= TObject(True);

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidth]
:= '164';

```



```

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft] :=
'width - 176';

ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop] :=
'-64';

    ExtraCaption['logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption]
:= ' <sha ;;0> <c>This is our logo<br> <c> <img>logo</img>';
    end;
end;
EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
    BeginUpdate();
    HTMLPicture['logo'] := 'E:\Exontrol\Exontrol.Logo\exontrol.logo.png';
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 10;
    with Layers.Item[OleVariant(9)] do
    begin
        RotateType := EXGAUGELib_TLB.exRotateBilinearInterpolation;
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        with Foreground do
        begin
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionAnchor] := OleVariant(2);
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWordWrap] :=
OleVariant(True);
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWidth] := '164';
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionLeft] := 'width - 176';
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionTop] := '-64';
            ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaption] := ' <sha ;;0> <c>This is
our logo<br> <c> <img>logo</img>';
        end;
    end;
end;

```

```

end;
EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.Object.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 10
with .Layers.Item(9)
.RotateType = 2
.OnDrag = 2
with .Foreground
.ExtraCaption("logo",3) = 2
.ExtraCaption("logo",8) = .T.
.ExtraCaption("logo",6) = "164"
.ExtraCaption("logo",4) = "width - 176"
.ExtraCaption("logo",5) = "-64"
.ExtraCaption("logo",0) = "<sha ;;0> <c> This is our logo<br> <c>
<img> logo </img>"
endwith
endwith
.EndUpdate
endwith

```

## dBASE Plus

```

local oGauge,var_Foreground,var_Layer

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.Template = [HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"] // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"

```

```
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Item(9)
var_Layer.RotateType = 2
var_Layer.OnDrag = 2
var_Foreground = var_Layer.Foreground
// var_Foreground.ExtraCaption("logo",3) = 2
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",3) = 2]
endwith
// var_Foreground.ExtraCaption("logo",8) = true
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",8) = True]
endwith
// var_Foreground.ExtraCaption("logo",6) = "164"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",6) = "164"]
endwith
// var_Foreground.ExtraCaption("logo",4) = "width - 176"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",4) = "width - 176"]
endwith
// var_Foreground.ExtraCaption("logo",5) = "-64"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",5) = "-64"]
```

```

endwith
// var_Foreground.ExtraCaption("logo",0) = "<sha ;;0><c>This is our logo<br>
<c><img>logo</img>"
with (oGauge)
    TemplateDef = [dim var_Foreground]
    TemplateDef = var_Foreground
    Template = [var_Foreground.ExtraCaption("logo",0) = "<sha ;;0><c>This is our
logo<br><c><img>logo</img>"]
endwith
oGauge.EndUpdate()

```

## XBasic (Alpha Five)

```

Dim oGauge as P
Dim var_Foreground as P
Dim var_Layer as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.Template = "HTMLPicture(`logo`) =
'E:\Exontrol\Exontrol.Logo\exontrol.logo.png" // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Item(9)
var_Layer.RotateType = 2
var_Layer.OnDrag = 2
var_Foreground = var_Layer.Foreground
' var_Foreground.ExtraCaption("logo",3) = 2
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,3) = 2"

' var_Foreground.ExtraCaption("logo",8) = .t.

```

```

oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,8) = True"

' var_Foreground.ExtraCaption("logo",6) = "164"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,6) = `164`"

' var_Foreground.ExtraCaption("logo",4) = "width - 176"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,4) = `width - 176`"

' var_Foreground.ExtraCaption("logo",5) = "-64"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,5) = `-64`"

' var_Foreground.ExtraCaption("logo",0) = "<sha ;;0> <c>This is our logo<br>
<c><img>logo</img>"
oGauge.TemplateDef = "dim var_Foreground"
oGauge.TemplateDef = var_Foreground
oGauge.Template = "var_Foreground.ExtraCaption(`logo`,0) = `<sha ;;0> <c>This
is our logo<br> <c><img>logo</img>`"

oGauge.EndUpdate()

```

## Visual Objects

```

local var_Foreground as IForeground
local var_Layer as ILayer

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:[HTMLPicture,"logo"] :=
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"

```

```

oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 10
var_Layer := oDCOCX_Exontrol1:Layers:[Item,9]
    var_Layer:RotateType := exRotateBilinearInterpolation
    var_Layer:OnDrag := exDoRotate
    var_Foreground := var_Layer:Foreground
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionAnchor] := 2
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionWordWrap] := true
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionWidth] := "164"
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionLeft] := "width - 176"
        var_Foreground:[ExtraCaption,"logo",exLayerCaptionTop] := "-64"
        var_Foreground:[ExtraCaption,"logo",exLayerCaption] := "<sha ;;0> <c>This is
our logo<br> <c> <img>logo</img>"
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge,var_Foreground,var_Layer

oGauge = ole_1.Object
oGauge.BeginUpdate()
oGauge.HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Item(9)
    var_Layer.RotateType = 2
    var_Layer.OnDrag = 2
    var_Foreground = var_Layer.Foreground
        var_Foreground.ExtraCaption("logo",3,2)
        var_Foreground.ExtraCaption("logo",8,true)
        var_Foreground.ExtraCaption("logo",6,"164")
        var_Foreground.ExtraCaption("logo",4,"width - 176")

```

```
var_Foreground.ExtraCaption("logo",5,"-64")
var_Foreground.ExtraCaption("logo",0,"<sha ;;0> <c>This is our logo<br> <c>
<img>logo</img>")
oGauge.EndUpdate()
```

## Visual DataFlex

### Procedure OnCreate

```
Forward Send OnCreate
Send ComBeginUpdate
Set ComHTMLPicture "logo" to "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers
Get ComLayers to voLayers
Handle hoLayers
Get Create (RefClass(cComLayers)) to hoLayers
Set pvComObject of hoLayers to voLayers
Set ComCount of hoLayers to 10
Send Destroy to hoLayers
Variant voLayers1
Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
Variant voLayer
Get ComItem of hoLayers1 9 to voLayer
Handle hoLayer
Get Create (RefClass(cComLayer)) to hoLayer
Set pvComObject of hoLayer to voLayer
Set ComRotateType of hoLayer to OLEexRotateBilinearInterpolation
Set ComOnDrag of hoLayer to OLEexDoRotate
Variant voForeground
Get ComForeground of hoLayer to voForeground
Handle hoForeground
```

```

Get Create (RefClass(cComForeground)) to hoForeground
Set pvComObject of hoForeground to voForeground
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionAnchor to
2
    Set ComExtraCaption of hoForeground "logo"
OLEexLayerCaptionWordWrap to True
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionWidth to
"164"
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionLeft to
"width - 176"
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaptionTop to
"-64"
    Set ComExtraCaption of hoForeground "logo" OLEexLayerCaption to "<sha
;;0> <c>This is our logo<br> <c> <img>logo</img> "
    Send Destroy to hoForeground
    Send Destroy to hoLayer
    Send Destroy to hoLayers1
    Send ComEndUpdate
End_Procedure

```

## XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge
    LOCAL oForeground
    LOCAL oLayer

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„ .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

```



```

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()

oGauge:SetProperty("HTMLPicture","logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.p

    oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 10
    oLayer := oGauge:Layers:Item(9)
        oLayer:RotateType := 2/*exRotateBilinearInterpolation*/
        oLayer:OnDrag := 2/*exDoRotate*/
        oForeground := oLayer:Foreground()

oForeground:SetProperty("ExtraCaption","logo",3/*exLayerCaptionAnchor*/,2)

oForeground:SetProperty("ExtraCaption","logo",8/*exLayerCaptionWordWrap*/,.T.)

oForeground:SetProperty("ExtraCaption","logo",6/*exLayerCaptionWidth*/,"164")

oForeground:SetProperty("ExtraCaption","logo",4/*exLayerCaptionLeft*/,"width -
176")

oForeground:SetProperty("ExtraCaption","logo",5/*exLayerCaptionTop*/,"-64")
    oForeground:SetProperty("ExtraCaption","logo",0/*exLayerCaption*/,"<sha
;;0> <c> This is our logo<br> <c> <img> logo</img>")
    oGauge:EndUpdate()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO

```



## property Foreground.Selectable as Boolean

Returns or sets a value that indicates whether all objects on the layer's foreground are selectable.

Type	Description
Boolean	A Boolean expression that specifies whether the layer's foreground is selectable or unselectable.

By default, the Selectable property is True. The Selectable property makes all layer' captions / extra-captions selectable or unselectable. The [Grayscale](#) property returns or sets a value that indicates whether the layer is show as grayscale. For instance, you can simulate a disabled layer by changing the layer's Grayscale property on True, and setting the layer's Selectable property on False. The [Visible](#) property shows or hides all layer's caption / extra-captions.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

# property Foreground.Visible as Boolean

Specifies if the objects of the layer's foreground are shown or hidden.

Type	Description
Boolean	A Boolean expression that specifies whether layer's foreground is visible or hidden.

By default, the Visible property is True. The Visible property shows or hides all layer's caption / extra-captions. The [Selectable](#) property makes all layer' captions / extra-captions selectable or unselectable.

Any of the following properties can be used to display a HTML caption:

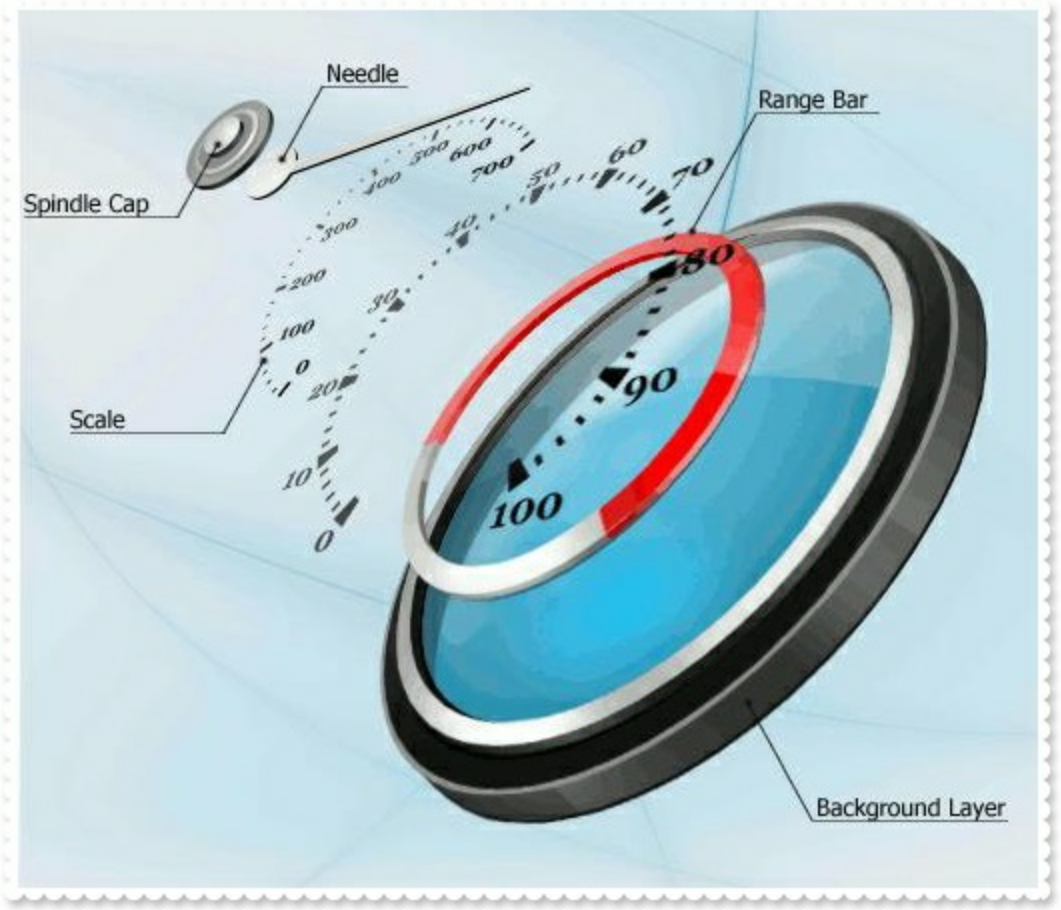
- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

# Gauge 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: {91628F12-393C-44EF-A558-83ED1790AAD3}. The object's program identifier is: "Exontrol.Gauge". The /COM object module is: "ExGauge.dll"

The eXGauge / eXLayers library provides graphics capabilities to visually display and edit the amount, level, or contents of something. The view can show one or more layers, where each layer can display one or more transparent pictures, HTML captions which can be clipped, moved, rotated or combination of them, by dragging the mouse, rolling the mouse wheel, or using the keyboard. Using the eXGauge / eXLayers library you can easily simulate any gauges, thermometers, meters, clocks, buttons, sliders, scales, knobs, dials, switches, progress, status, indicators, LEDs, and so on. As usual, there are no dependencies to MFC, VB, VCL, or anything else.

The following screen shot shows the idea of having multiple layers that are transparent and you can adjust the offset of the images, the transparency levels and rotate each image to create different types of gauges simple by changing the layer graphics:



The following table shows how you can create / access different type of objects ( red items indicates the name of the property/method ):

EXGAUGELib.Gauge
"Layers" -> EXGAUGELib.Layers

"VisualAppearance" -> EXGAUGELib.Appearance

EXGAUGELib.Layers

"Add(Variant)" -> EXGAUGELib.Layer

"Item(Variant)" -> EXGAUGELib.Layer

"VisibleItem(Long)" -> EXGAUGELib.Layer

EXGAUGELib.Layer

"Background" -> EXGAUGELib.Background

"Clip" -> EXGAUGELib.Clip

"Foreground" -> EXGAUGELib.Foreground

EXGAUGELib.Background

"Color" -> EXGAUGELib.Color

"ExtraPicture(Variant)" -> EXGAUGELib.Picture

"Picture" -> EXGAUGELib.Picture

EXGAUGELib.Clip

"Ellipse" -> EXGAUGELib.ClipEllipse

"Picture" -> EXGAUGELib.ClipPicture

"Pie" -> EXGAUGELib.ClipPie

"Polygon" -> EXGAUGELib.ClipPolygon

"Rectangle" -> EXGAUGELib.ClipRectangle

"RoundRectangle" -> EXGAUGELib.ClipRoundRectangle

The following table shows how you can create / access different type of objects ( red items indicates the name of the property/method ):

EXGAUGELib.Appearance <- "VisualAppearance" of EXGAUGELib.Gauge

EXGAUGELib.Background <- "Background" of EXGAUGELib.Layer

EXGAUGELib.Clip <- "Clip" of EXGAUGELib.Layer

EXGAUGELib.ClipEllipse <- "Ellipse" of EXGAUGELib.Clip

EXGAUGELib.ClipPicture <- "Picture" of EXGAUGELib.Clip

EXGAUGELib.ClipPie <- "Pie" of EXGAUGELib.Clip

EXGAUGELib.ClipPolygon <- "Polygon" of EXGAUGELib.Clip

EXGAUGELib.ClipRectangle <- "Rectangle" of EXGAUGELib.Clip

EXGAUGELib.ClipRoundRectangle <- "RoundRectangle" of EXGAUGELib.Clip

EXGAUGELib.Color <- "Color" of EXGAUGELib.Background

EXGAUGELib.Foreground <- "Foreground" of EXGAUGELib.Layer

EXGAUGELib.Layer <- "Add(Variant)" of EXGAUGELib.Layers

EXGAUGELib.Layer <- "Item(Variant)" of EXGAUGELib.Layers

EXGAUGELib.Layer <- "VisibleItem(Long)" of EXGAUGELib.Layers

EXGAUGELib.Layers <- "Layers" of EXGAUGELib.Gauge  
EXGAUGELib.Picture <- "ExtraPicture(Variant)" of EXGAUGELib.Background  
EXGAUGELib.Picture <- "Picture" of EXGAUGELib.Background

The Gauge object supports the following properties and methods:

Name	Description
<a href="#">AllowCopyTemplate</a>	Specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.
<a href="#">AllowMoveOnClick</a>	Allows moving the window that contains the control to a new position, as you would do by clicking the form's title/caption.
<a href="#">AllowSmoothChange</a>	Specifies the properties of the layers that support smooth change.
<a href="#">AnchorFromPoint</a>	Retrieves the identifier of the anchor from point.
<a href="#">Appearance</a>	Retrieves or sets the control's appearance.
<a href="#">AttachTemplate</a>	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
<a href="#">BackColor</a>	Specifies the control's background color.
<a href="#">Background</a>	Returns or sets a value that indicates the background color for parts in the control.
<a href="#">BeginUpdate</a>	Maintains performance when items are added to the control one at a time. This method prevents the control from painting until the EndUpdate method is called.
<a href="#">Caption</a>	Specifies the caption on the control.
<a href="#">CopyTo</a>	Exports the control's view to an EMF file.
<a href="#">Debug</a>	Displays the control in debug mode.
<a href="#">DefaultLayer</a>	Defines the default value for properties of the layers to be created.
<a href="#">Enabled</a>	Enables or disables the control.
<a href="#">EndUpdate</a>	Resumes painting the control after painting is suspended by the BeginUpdate method.
<a href="#">EventParam</a>	Retrieves or sets a value that indicates the current's event parameter.
<a href="#">ExecuteTemplate</a>	Executes a template and returns the result.

<a href="#">ExtraCaption</a>	Specifies any extra caption on the control.
<a href="#">Font</a>	Retrieves or sets the control's font.
<a href="#">ForeColor</a>	Specifies the control's foreground color.
<a href="#">FormatABC</a>	Formats the A,B,C values based on the giving expression and returns the result.
<a href="#">FormatAnchor</a>	Specifies the visual effect for anchor elements in HTML captions.
<a href="#">FreezeEvents</a>	Prevents the control to fire any event.
<a href="#">HTMLPicture</a>	Adds or replaces a picture in HTML captions.
<a href="#">hWnd</a>	Retrieves the control's window handle.
<a href="#">Images</a>	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
<a href="#">ImageSize</a>	Retrieves or sets the size of icons the control displays..
<a href="#">LayerAutoSize</a>	Specifies the index of the layer that determines the size to display all layers.
<a href="#">LayerClipTo</a>	Specifies the index of the layer that clips the entire control to.
<a href="#">LayerClipToParent</a>	Indicates if the LayerClipTo method clips the control itself, parent or the owner of the control.
<a href="#">LayerDragAny</a>	Specifies the index of the layer to drag (rotate or move) once the user clicks anywhere on the control.
<a href="#">LayerFromPoint</a>	Retrieves the index of the layer from the point ( only visible and selectable objects are included ).
<a href="#">LayerOfValue</a>	Specifies the index of the layer whose value represents the control's Value property.
<a href="#">Layers</a>	Returns the Layers collection.
<a href="#">LayerUpdate</a>	Specifies where the control updates its content.
<a href="#">PicturesName</a>	Specifies the expression that indicates the name of the picture to be loaded on each layer.
<a href="#">PicturesPath</a>	Specifies the path to load the pictures from.
<a href="#">Refresh</a>	Refreses the control.
<a href="#">Replacelcon</a>	Adds a new icon, replaces an icon or clears the control's image list.
<a href="#">ShowImageList</a>	Specifies whether the control's image list window is visible or hidden.



<a href="#">ShowLayers</a>	Indicates the only layers to be shown on the control.
<a href="#">ShowToolTip</a>	Shows the specified tooltip at given position.
<a href="#">Template</a>	Specifies the control's template.
<a href="#">TemplateDef</a>	Defines inside variables for the next Template/ExecuteTemplate call.
<a href="#">TemplatePut</a>	Defines inside variables for the next Template/ExecuteTemplate call.
<a href="#">TimerInterval</a>	Returns or sets the number of milliseconds between calls of control's Timer event.
<a href="#">ToolTipDelay</a>	Specifies the time in ms that passes before the ToolTip appears.
<a href="#">ToolTipFont</a>	Retrieves or sets the tooltip's font.
<a href="#">ToolTipPopDelay</a>	Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.
<a href="#">ToolTipWidth</a>	Specifies a value that indicates the width of the tooltip window, in pixels.
<a href="#">ToTemplate</a>	Generates the control's template.
<a href="#">TransparentColorFrom</a>	Specifies the transparent color for all pictures in all layers, to define transparency part (from).
<a href="#">TransparentColorTo</a>	Specifies the transparent color for all pictures in all layers, to define transparency part (to).
<a href="#">Value</a>	Specifies the control's value.
<a href="#">Version</a>	Retrieves the control's version.
<a href="#">VisualAppearance</a>	Retrieves the control's appearance.

# property Gauge.AllowCopyTemplate as Boolean

Specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.

Type	Description
Boolean	A Boolean expression that specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.

By default, AllowCopyTemplate property is True ( for evaluation version ), and False ( for registered version ). The AllowCopyTemplate property specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form. The AllowCopyTemplate property is available for /COM version only, and it was provided for a simple way of copying the control's content to template form, no matter of your programming language. The property uses the [ToTemplate](#) property to generate the control's template, at runtime. The format of the clipboard being copied is plain text. Use the [Template](#) property to apply the generated template to an empty control.

## property Gauge.AllowMoveOnClick as Boolean

Allows moving the window that contains the control to a new position, as you would do by clicking the form's title/caption.

Type	Description
Boolean	A Boolean expression that specifies whether the user can move the control on the screen by clicking it.

By default, the AllowMoveOnClick property is False. The AllowMoveOnClick property allows moving the window that contains the control to a new position, as you would do by clicking the form's title/caption. For instance, you can use the AllowMoveOnClick property on True, when you create a widget to be displayed on the screen, and so the user can move the widget by clicking it.

You can use any of the following to convert your control to a widget:

- The [LayerClipTo](#) property specifies the index of the layer that clips the entire control to.
- The [LayerUpdate](#) property indicates where the control's content is updated.

# property Gauge.AllowSmoothChange as SmoothPropertyEnum

Specifies the properties of the layers that support smooth change.

Type	Description
<a href="#">SmoothPropertyEnum</a>	A SmoothPropertyEnum expression the properties of the layer that can be changed gradually.

By default, the AllowSmoothChange property is exLayerTransparency | exLayerBrightness | exLayerContrast. Use the AllowSmoothChange property to disable changing gradually any brightness / contrast or the transparency, of the layer. For instance, a gradually change means that you can change the layer's transparency from 0 to 50 in a short time, with intermediate values ( smooth change ).

The AllowSmoothChange property changes gradually one / or more properties like follow:

- [Brightness](#), Specifies the percent of brightness to apply to the layer.
- [Contrast](#), Specifies the percent of contrast to apply to the layer.
- [Transparency](#), Gets or sets a value that indicates percent of the transparency to display the layer.

The [MouseDown](#) / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects. The AllowSmoothChange property specifies the properties of the layers that support smooth change. For instance, you can use the MouseIn / MouseOut event to change gradually the brightness / contrast or the transparency, of the layer, while the AllowSmoothChange property is not exSmoothChangeless.

# property Gauge.AnchorFromPoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS) as String

Retrieves the identifier of the anchor from 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 Y location of the mouse pointer. The y values is always expressed in client coordinates.
String	A String expression that specifies the identifier (id) of the anchor element from the point, or empty string if there is no anchor element at the cursor.

Use the AnchorFromPoint property to determine the identifier of the anchor from the point. Use the <a id;options> anchor elements to add hyperlinks to cell's caption. The control fires the [AnchorClick](#) event when the user clicks an anchor element. Use the [ShowToolTip](#) method to show the specified tooltip at given or cursor coordinates. The [MouseMove](#) event is generated continually as the mouse pointer moves across the control.

# property Gauge.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
<a href="#">AppearanceEnum</a>	An AppearanceEnum expression that indicates the control's appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the <a href="#">Appearance</a> 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.

Use the Appearance property to specify the control's border.

## method Gauge.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 Gauge1_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 Gauge.BackColor as Color

Specifies the control's background color.

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

The BackColor property specifies the control's background color. The [ForeColor](#) property specifies the control's foreground color.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows an extra-caption associated with the layer:



# property Gauge.Background(Part as BackgroundPartEnum) as Color

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

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

The Background property specifies a background color or a visual appearance for specific parts in the control. If the Background property is 0, the control draws the part as default. Use the [Add](#) method to add new skins to the control. Use the [Remove](#) method to remove a specific skin from the control. Use the [Clear](#) method to remove all skins in the control. Use the [BeginUpdate](#) and [EndUpdate](#) methods to maintain performance while init the control. Use the [Refresh](#) method to refresh the control.

For instance:

- Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips
- Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color
- Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

Use the [ShowToolTip](#) method to display a custom tooltip. The [ToolTip](#) / [ToolTipTitle](#) property indicates the layer's tooltip. The [LayerFromPoint](#) property returns the index of the layer from the cursor. Use the [ToolTipWidth](#) property to specify the width of the tooltip window Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipFont](#) property to change the tooltip's font.

# method Gauge.BeginUpdate ()

Maintains performance when items are added to the control one at a time. This method prevents the control from painting until the EndUpdate method is called.

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

# Property Gauge.Caption(Property as PropertyLayerCaptionEnum) as Variant

Specifies the caption on the control.

Type	Description
Property as <a href="#">PropertyLayerCaptionEnum</a>	A PropertyLayerCaptionEnum expression that specifies the caption's property to be changed.
Variant	A VARIANT expression that specifies the value of the caption's property.

The control support unlimited HTML captions to be place anywhere on the control or on any layer of the control. The Caption(exLayerCaption) specifies the HTML caption to be shown on the control/layer. The [Images](#) method specifies the list of icons the control can display. The [HTMLPicture](#) adds or replaces a picture in HTML captions. The Caption(exLayerCaptionBackgroundExt) property indicates unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the control / layer's background. The caption on the control stay on its position, no matter what layer is moved or rotated, while a caption on a layer gets moved or rotated or clipped together with the layer itself. The [Visible](#) property shows or hides all layer's caption / extra-captions. The [LayerToClientX](#) / [LayerToClientY](#) properties translate a point from the layer ( as it is moved or rotated ) to the control's view.

Any of the following properties can be used to display a HTML caption:

- Caption property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows a caption on the Top-Left side of the control, and one extra caption to to Bottom-Right side of the control:



The following samples show how you can place caption on the Top-Left side of the control, and one extra caption to to Bottom-Right side of the control:

### VBA (MS Access, Excell...)

With Gauge1

```
.BeginUpdate
.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.Caption(0) = "This is just a caption"
.ExtraCaption("logo",3) = 2
.ExtraCaption("logo",8) = True
.ExtraCaption("logo",6) = "164"
.ExtraCaption("logo",4) = "width - 164"
.ExtraCaption("logo",0) = "<c>This is our logo<br><c> <img>logo</img>"
.EndUpdate
End With
```

### VB6

With Gauge1

```
.BeginUpdate
.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
```

```

1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.Caption(exLayerCaption) = "This is just a caption"
.ExtraCaption("logo",exLayerCaptionAnchor) = 2
.ExtraCaption("logo",exLayerCaptionWordWrap) = True
.ExtraCaption("logo",exLayerCaptionWidth) = "164"
.ExtraCaption("logo",exLayerCaptionLeft) = "width - 164"
.ExtraCaption("logo",exLayerCaption) = "<c>This is our logo<br> <c>
<img>logo</img>"
.EndUpdate
End With

```

## VB.NET

```

With Exgauge1
.BeginUpdate()
.set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5

.set_Caption(exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"This
is just a caption")

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
- 164")

```

```
.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
<c>This is our logo<br><c><img>logo</img>")
.EndUpdate()
End With
```

## VB.NET for /COM

```
With AxGauge1
.BeginUpdate()
.set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"This is just
a caption")

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnch

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWor

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidt

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft,'
- 164")
.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"
<c>This is our logo<br><c><img>logo</img>")
.EndUpdate()
End With
```

## C++

```
/*
Copy and paste the following directives to your header file as
```

*it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control Library'*

```
#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1-
> PutHTMLPicture(L"logo", "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
spGauge1-> PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
spGauge1-> PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1-> GetLayers()-> PutCount(5);
spGauge1-> PutCaption(EXGAUGELib::exLayerCaption, "This is just a caption");
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionAnchor, long(2));
spGauge1-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWordWrap, VARIANT_TRUE);
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWidth, "164");
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionLeft, "width - 164");
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaption, "<c>This is our
logo<br><c><img>logo</img>");
spGauge1-> EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();
Gauge1->HTMLPicture[L"logo"] =
TVariant("E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`;
Gauge1->Layers->Count = 5;
Gauge1-> Caption[Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] =
TVariant("This is just a caption");
```





```
<c>This is our logo<br><c><img>logo</img>");  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"></OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.BeginUpdate();  
    Gauge1.HTMLPicture("logo") = "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png";  
    Gauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 5;  
    Gauge1.Caption(0) = "This is just a caption";  
    Gauge1.ExtraCaption("logo",3) = 2;  
    Gauge1.ExtraCaption("logo",8) = true;  
    Gauge1.ExtraCaption("logo",6) = "164";  
    Gauge1.ExtraCaption("logo",4) = "width - 164";  
    Gauge1.ExtraCaption("logo",0) = "<c>This is our logo<br><c>  
<img>logo</img>";  
    Gauge1.EndUpdate();  
}  
</SCRIPT>  
</BODY>
```

## VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"></OBJECT>
```

```

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
        .PicturesName = "`Layer` + int(value + 1) + `.png`"
        .Layers.Count = 5
        .Caption(0) = "This is just a caption"
        .ExtraCaption("logo",3) = 2
        .ExtraCaption("logo",8) = True
        .ExtraCaption("logo",6) = "164"
        .ExtraCaption("logo",4) = "width - 164"
        .ExtraCaption("logo",0) = "<c>This is our logo<br><c><img>logo</img>"
        .EndUpdate
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 5;
axGauge1.set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"This
is just a caption");
axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa

axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa

axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa

```

```

axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
- 164");
axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
<c>This is our logo<br> <c> <img>logo</img>");
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    ;

    super();

    exgauge1.BeginUpdate();
    exgauge1.HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(5);
    exgauge1.Caption(0/*exLayerCaption*/, "This is just a caption");

    exgauge1.ExtraCaption("logo",3/*exLayerCaptionAnchor*/,COMVariant::createFromInt(

    exgauge1.ExtraCaption("logo",8/*exLayerCaptionWordWrap*/,COMVariant::createFrom

    exgauge1.ExtraCaption("logo",6/*exLayerCaptionWidth*/,"164");
    exgauge1.ExtraCaption("logo",4/*exLayerCaptionLeft*/,"width - 164");
    exgauge1.ExtraCaption("logo",0/*exLayerCaption*/,"<c>This is our logo<br> <c>
<img>logo</img>");
    exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
  BeginUpdate();
  set_HTMLPicture('logo','E:\Exontrol\Exontrol.Logo\exontrol.logo.png');
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,'This is just a
caption');

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnch

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWord'

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidth

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft,'w
- 164');

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,'<c>T
is our logo<br><c> <img>logo</img>');
  EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
  BeginUpdate();
  HTMLPicture['logo'] := 'E:\Exontrol\Exontrol.Logo\exontrol.logo.png';
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
  PicturesName := 'Layer` + int(value + 1) + `.png`;

```

```

Layers.Count := 5;
Caption[EXGAUGELib_TLB.exLayerCaption] := 'This is just a caption';
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionAnchor] := OleVariant(2);
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWordWrap] :=
OleVariant(True);
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWidth] := '164';
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionLeft] := 'width - 164';
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaption] := '<c>This is our logo<br>
<c> <img>logo</img>';
EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.Object.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.Object.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.Object.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Object.Layers.Count = 5
.Object.Caption(0) = "This is just a caption"
.Object.ExtraCaption("logo",3) = 2
.Object.ExtraCaption("logo",8) = .T.
.Object.ExtraCaption("logo",6) = "164"
.Object.ExtraCaption("logo",4) = "width - 164"
.Object.ExtraCaption("logo",0) = "<c>This is our logo<br> <c>
<img>logo</img>"
.Object.EndUpdate
endwith

```

## dBASE Plus

local oGauge

```

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.Template = [HTMLPicture("logo") =

```

```

"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"] // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.Template = [Caption(0) = "This is just a caption"] // oGauge.Caption(0) =
"This is just a caption"
oGauge.Template = [ExtraCaption("logo",3) = 2] // oGauge.ExtraCaption("logo",3) = 2
oGauge.Template = [ExtraCaption("logo",8) = True] // oGauge.ExtraCaption("logo",8)
= true
oGauge.Template = [ExtraCaption("logo",6) = "164"] // oGauge.ExtraCaption("logo",6)
= "164"
oGauge.Template = [ExtraCaption("logo",4) = "width - 164"] //
oGauge.ExtraCaption("logo",4) = "width - 164"
oGauge.Template = [ExtraCaption("logo",0) = "<c>This is our logo<br><c>
<img>logo</img>"] // oGauge.ExtraCaption("logo",0) = "<c>This is our logo<br>
<c><img>logo</img>"
oGauge.EndUpdate()

```

## XBasic (Alpha Five)

```

Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.Template = "HTMLPicture(`logo`) =
`E:\Exontrol\Exontrol.Logo\exontrol.logo.png`" // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.Template = "Caption(0) = `This is just a caption`" // oGauge.Caption(0) =
"This is just a caption"
oGauge.Template = "ExtraCaption(`logo`,3) = 2" // oGauge.ExtraCaption("logo",3) = 2

```

```

oGauge.Template = "ExtraCaption('logo`,8) = True" // oGauge.ExtraCaption("logo",8)
= .t.
oGauge.Template = "ExtraCaption('logo`,6) = `164`" // oGauge.ExtraCaption("logo",6)
= "164"
oGauge.Template = "ExtraCaption('logo`,4) = `width - 164`" //
oGauge.ExtraCaption("logo",4) = "width - 164"
oGauge.Template = "ExtraCaption('logo`,0) = `

```

## Visual Objects

```

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:[HTMLPicture,"logo"] :=
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 5
oDCOCX_Exontrol1:[Caption,exLayerCaption] := "This is just a caption"
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionAnchor] := 2
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionWordWrap] := true
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionWidth] := "164"
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionLeft] := "width - 164"
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaption] := "<c>This is our
logo<br><c><img>logo</img>"
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge

oGauge = ole_1.Object
oGauge.BeginUpdate()

```



```

oGauge.HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.Caption(0,"This is just a caption")
oGauge.ExtraCaption("logo",3,2)
oGauge.ExtraCaption("logo",8,true)
oGauge.ExtraCaption("logo",6,"164")
oGauge.ExtraCaption("logo",4,"width - 164")
oGauge.ExtraCaption("logo",0,"<c>This is our logo<br><c><img>logo</img>")
oGauge.EndUpdate()

```

## Visual DataFlex

```

Procedure OnCreate
  Forward Send OnCreate
  Send ComBeginUpdate
  Set ComHTMLPicture "logo" to "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
  Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
  Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
  Variant voLayers
  Get ComLayers to voLayers
  Handle hoLayers
  Get Create (RefClass(cComLayers)) to hoLayers
  Set pvComObject of hoLayers to voLayers
    Set ComCount of hoLayers to 5
  Send Destroy to hoLayers
  Set ComCaption OLEexLayerCaption to "This is just a caption"
  Set ComExtraCaption "logo" OLEexLayerCaptionAnchor to 2
  Set ComExtraCaption "logo" OLEexLayerCaptionWordWrap to True
  Set ComExtraCaption "logo" OLEexLayerCaptionWidth to "164"
  Set ComExtraCaption "logo" OLEexLayerCaptionLeft to "width - 164"
  Set ComExtraCaption "logo" OLEexLayerCaption to "<c>This is our logo<br><c>
<img>logo</img>"

```

```
Send ComEndUpdate
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480} ,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()

    oGauge:SetProperty("HTMLPicture","logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")

    oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 5
    oGauge:SetProperty("Caption",0/*exLayerCaption*/,"This is just a caption")
    oGauge:SetProperty("ExtraCaption","logo",3/*exLayerCaptionAnchor*/2)
    oGauge:SetProperty("ExtraCaption","logo",8/*exLayerCaptionWordWrap*/,.T.)
    oGauge:SetProperty("ExtraCaption","logo",6/*exLayerCaptionWidth*/,"164")
    oGauge:SetProperty("ExtraCaption","logo",4/*exLayerCaptionLeft*/,"width -
164")
```

```
oGauge:SetProperty("ExtraCaption","logo",0/*exLayerCaption*/,"<c>This is our  
logo<br><c><img>logo</img>" )
```

```
oGauge:EndUpdate()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

# property Gauge.CopyTo (File as String) as Variant

Exports the control's view to an EMF file.

Type	Description
File as String	<p>A String expression that indicates the name of the file to be saved. If present, the CopyTo property retrieves True, if the operation succeeded, else False it is failed. If the File parameter is missing or empty, the CopyTo property retrieves an one dimension safe array of bytes that contains the EMF content.</p> <p>If the File parameter is not empty, the extension ( characters after last dot ) determines the graphical/ format of the file to be saved as follows:</p> <ul style="list-style-type: none"><li>• <b>*.bmp *.dib *.rle</b>, saves the control's content in <b>BMP</b> format.</li><li>• <b>*.jpg *.jpe *.jpeg *.jfif</b>, saves the control's content in <b>JPEG</b> format.</li><li>• <b>*.gif</b>, , saves the control's content in <b>GIF</b> format.</li><li>• <b>*.tif *.tiff</b>, saves the control's content in <b>TIFF</b> format.</li><li>• <b>*.png</b>, saves the control's content in <b>PNG</b> format.</li><li>• <b>*.pdf</b>, saves the control's content to PDF format. The File argument may carry up to 4 parameters separated by the   character in the following order: <b><i>filename.pdf   paper size   margins   options</i></b>. In other words, you can specify the file name of the PDF document, the paper size, the margins and options to build the PDF document. By default, the paper size is 210 <b>mm</b> × 297 <b>mm</b> ( A4 format ) and the margins are 12.7 <b>mm</b> 12.7 <b>mm</b> 12.7 <b>mm</b> 12.7 <b>mm</b>. The units for the paper size and margins can be <b>pt</b> for PostScript Points, <b>mm</b> for Millimeters, <b>cm</b> for Centimeters, <b>in</b> for Inches and <b>px</b> for pixels. If PostScript Points are used if unit is missing. For instance, 8.27 in x 11.69 in, indicates the size of the paper in inches. Currently, the options can be <b>single</b>, which indicates that the control's content is exported to a single PDF page. For instance, the CopyTo("shot.pdf 33.11 in x 46.81 in 0 0 0 0 single") exports the control's content to an A0 single PDF page, with no margins.</li><li>• <b>*.emf</b> or any other extension determines the control to</li></ul>

save the control's content in **EMF** format.

For instance, the `CopyTo("c:\temp\snapshot.png")` property saves the control's content in PNG format to `snapshot.png` file.

---

Variant

A boolean expression that indicates whether the File was successful saved, if the File parameter is not empty, or a one dimension safe array of bytes, if the File parameter is empty string.

---

The `CopyTo` method copies/exports the control's view to BMP, PNG, JPG, GIF, TIFF, PDF or EMF graphical files.

- The **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- The **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- The **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- The **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.
- The **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- The **PDF** (Portable Document Format) is a file format used to present documents in a manner independent of application software, hardware, and operating systems. Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it.
- The **EMF** ( Enhanced Metafile Format ) is a 32-bit format that can contain both vector information and bitmap information. This format is an improvement over the Windows Metafile Format and contains extended features, such as the following

Built-in scaling information

Built-in descriptions that are saved with the file

Improvements in color palettes and device independence

The EMF format is an extensible format, which means that a programmer can modify the original specification to add functionality or to meet specific needs. You can paste this format to Microsoft Word, Excel, Front Page, Microsoft Image Composer and any application that know to handle EMF formats.

The following VB sample saves the control's content to a EMF file:

```
If (Control.CopyTo("c:\temp\test.emf")) Then
    MsgBox "test.emf file created, open it using the mspaint editor."
End If
```

The following VB sample prints the EMF content ( as bytes, File parameter is empty string ):

```
Dim i As Variant
For Each i In Control.CopyTo("")
    Debug.Print i
Next
```

# property Gauge.Debug as DebugLayerEnum

Displays the control in debug mode.

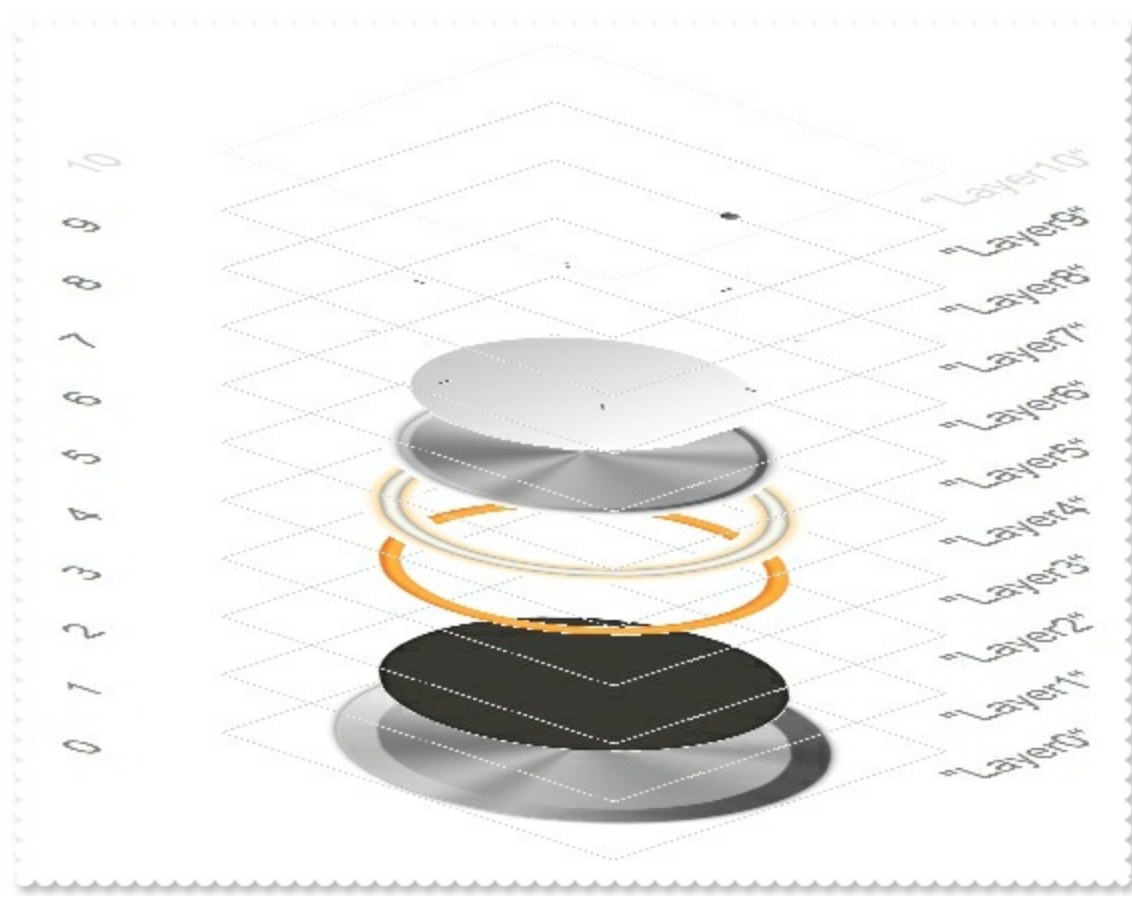
Type	Description
<a href="#">DebugLayerEnum</a>	A DebugLayerEnum expression that indicates whether the control displays layers in debug mode.

By default, the Debug property is exNoDebugLayer. Use the Debug property to display the layers in debug mode. The [ShowLayers](#) property indicates the only layers to be shown on the control. The [Debug](#) property specifies debugging information to be shown while dragging the layers. Also, properties like [OnDrag](#), [LayerFromPoint](#) are not valid while the control is running in debug mode.

The following screen shot shows the control using pictures from the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob folder ( Debug property is exNoDebugLayer, default ):



The following screen shot shows the control while Debug property is exDebugLayers:





# property Gauge.DefaultLayer(Property as DefaultLayerPropertyEnum) as Variant

Defines the default value for properties of the layers to be created.

Type	Description
Property as <a href="#">DefaultLayerPropertyEnum</a>	A DefaultLayerPropertyEnum expression that specifies the property to change the default value
Variant	A VARIANT expression that specifies the property's default value.

The DefaultLayer property defines the default value for properties of the layers to be created. Any call of the DefaultLayer property has effect for any new layer added to the control's collection. Changing the DefaultLayer property does not have any effect on already existing layers.

The following samples show how you can load all layers with a semi-transparency (50%):

## VBA (MS Access, Excell...)

```
With Gauge1
    .AllowSmoothChange = 0
    .DefaultLayer(22) = 50
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
End With
```

## VB6

```
With Gauge1
    .AllowSmoothChange = exSmoothChangeless
    .DefaultLayer(exDefLayerTransparency) = 50
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
End With
```

## VB.NET

With Exgauge1

```
.AllowSmoothChange =  
exontrol.EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless  
  
.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerTranspar  
  
.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
.PicturesName = "`Layer` + int(value + 1) + `.png`"  
.Layers.Count = 11  
End With
```

## VB.NET for /COM

With AxGauge1

```
.AllowSmoothChange = EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless  
  
.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerTransparency,50)  
  
.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
.PicturesName = "`Layer` + int(value + 1) + `.png`"  
.Layers.Count = 11  
End With
```

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control  
    Library'  
  
    #import <ExGauge.dll>  
    using namespace EXGAUGELib;  
*/  
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-  
>GetControlUnknown();
```

```

spGauge1->PutAllowSmoothChange(EXGAUGELib::exSmoothChangeless);
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerTransparency,long(50));
spGauge1->PutPicturesPath(L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(11);

```

## C++ Builder

```

Gauge1->AllowSmoothChange =
ExgaugeLib_tlb::SmoothPropertyEnum::exSmoothChangeless;
Gauge1-
>DefaultLayer[ExgaugeLib_tlb::DefaultLayerPropertyEnum::exDefLayerTransparency] =
TVariant(50);
Gauge1->PicturesPath = L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`;
Gauge1->Layers->Count = 11;

```

## C#

```

exgauge1.AllowSmoothChange =
exontrol.EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;
exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayer

exgauge1.PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
exgauge1.PicturesName = "Layer` + int(value + 1) + `.png`;
exgauge1.Layers.Count = 11;

```

## JScript/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"></OBJECT>

```

```

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.AllowSmoothChange = 0;
    Gauge1.DefaultLayer(22) = 50;
    Gauge1.PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    Gauge1.Layers.Count = 11;
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .AllowSmoothChange = 0
        .DefaultLayer(22) = 50
        .PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob"
        .PicturesName = "`Layer` + int(value + 1) + `.png`"
        .Layers.Count = 11
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.AllowSmoothChange =
EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;
axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerTranspa

axGauge1.PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = ""Layer` + int(value + 1) + `.png`;
axGauge1.Layers.Count = 11;

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    ;

    super();

    exgauge1.AllowSmoothChange(0/*exSmoothChangeless*/);

    exgauge1.DefaultLayer(22/*exDefLayerTransparency*/,COMVariant::createFromInt(50));

    exgauge1.PicturesPath("C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName(""Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(11);
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
    AllowSmoothChange := EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;

    set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerTransparency,TOB

    PicturesPath := 'C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob';

```

```

PicturesName := ``Layer` + int(value + 1) + `.png`;
Layers.Count := 11;
end

```

## Delphi (standard)

```

with Gauge1 do
begin
    AllowSmoothChange := EXGAUGELib_TLB.exSmoothChangeless;
    DefaultLayer[EXGAUGELib_TLB.exDefLayerTransparency] := OleVariant(50);
    PicturesPath := 'C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := ``Layer` + int(value + 1) + `.png`;
    Layers.Count := 11;
end

```

## VFP

```

with thisform.Gauge1
    .AllowSmoothChange = 0
    .Object.DefaultLayer(22) = 50
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = ``Layer` + int(value + 1) + `.png`
    .Layers.Count = 11
endwith

```

## dBASE Plus

```

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.AllowSmoothChange = 0
oGauge.Template = [DefaultLayer(22) = 50] // oGauge.DefaultLayer(22) = 50
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = ``Layer` + int(value + 1) + `.png`
oGauge.Layers.Count = 11

```

## XBasic (Alpha Five)

```
Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.AllowSmoothChange = 0
oGauge.Template = "DefaultLayer(22) = 50" // oGauge.DefaultLayer(22) = 50
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
```

## Visual Objects

```
oDCOCX_Exontrol1:AllowSmoothChange := exSmoothChangeless
oDCOCX_Exontrol1:[DefaultLayer,exDefLayerTransparency] := 50
oDCOCX_Exontrol1:PicturesPath := "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers.Count := 11
```

## PowerBuilder

```
OleObject oGauge

oGauge = ole_1.Object
oGauge.AllowSmoothChange = 0
oGauge.DefaultLayer(22,50)
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
```

## Visual DataFlex

### Procedure OnCreate

Forward Send OnCreate

Set ComAllowSmoothChange to OLEexSmoothChangeless

Set ComDefaultLayer OLEexDefLayerTransparency to 50

Set ComPicturesPath to "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"

Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Set **ComCount** of hoLayers to 11

Send Destroy to hoLayers

End\_Procedure

## XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

### PROCEDURE Main

```
LOCAL oForm
```

```
LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
LOCAL oGauge
```

```
oForm := XbpDialog():new( AppDesktop() )
```

```
oForm:drawingArea:clipChildren := .T.
```

```
oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oGauge := XbpActiveXControl():new( oForm:drawingArea )
```

```
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-  
83ED1790AAD3}*/
```

```
oGauge:create(,, {10,60},{610,370} )
```



```
oGauge:AllowSmoothChange := 0/*exSmoothChangeless*/  
oGauge:SetProperty("DefaultLayer",22/*exDefLayerTransparency*/,50)  
oGauge:PicturesPath := "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"  
oGauge:Layers():Count := 11
```

```
oForm:Show()  
DO WHILE nEvent != xbeP_Quit  
    nEvent := AppEvent( @mp1, @mp2, @oXbp )  
    oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

# property Gauge.Enabled as Boolean

Enables or disables the control.

Type	Description
Boolean	A boolean expression that determines whether an control can respond to user-generated events.

By default, the Enabled property is True. The Enabled property specifies whether the entire control is enabled or disabled. The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. You can use the [Grayscale](#) property to show the entire layer in gray scale ( disable state). For instance, you can simulate a disabled layer by changing the layer's Grayscale property on True, and setting the layer's Selectable property on False.

# method Gauge.EndUpdate ()

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

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

# property Gauge.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 Gauge.ExecuteTemplate (Template as String)

Executes a template and returns the result.

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

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

For instance, the following sample retrieves the control's background color:

```
Debug.Print Gauge1.ExecuteTemplate("BackColor")
```

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

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

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

The Template 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 Gauge.ExtraCaption(Key as Variant, Property as PropertyLayerCaptionEnum) as Variant

Specifies any extra caption on the control.

Type	Description
Key as Variant	A VARIANT expression that specifies the key of the extra caption. You can use any value to identify one extra caption.
Property as <a href="#">PropertyLayerCaptionEnum</a>	A PropertyLayerCaptionEnum expression that specifies the extra caption's property to be changed.
Variant	A VARIANT expression that specifies the value of the extra caption's property.

The control support unlimited HTML captions to be place anywhere on the control or on any layer of the control. The Caption( exLayerCaption) specifies the HTML caption to be shown on the control/layer. The [Images](#) method specifies the list of icons the control can display. The [HTMLPicture](#) adds or replaces a picture in HTML captions. The Caption(exLayerCaptionBackgroundExt) property indicates unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the control / layer's background. The caption on the control stay on its position, no matter what layer is moved or rotated, while a caption on a layer gets moved or rotated together with the layer itself.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- ExtraCaption property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows a caption on the Top-Left side of the control, and one extra caption to to Bottom-Right side of the control:



The following samples show how you can place caption on the Top-Left side of the control, and one extra caption to to Bottom-Right side of the control:

### VBA (MS Access, Excell...)

With Gauge1

```
.BeginUpdate
.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.Caption(0) = "This is just a caption"
.ExtraCaption("logo",3) = 2
.ExtraCaption("logo",8) = True
.ExtraCaption("logo",6) = "164"
.ExtraCaption("logo",4) = "width - 164"
.ExtraCaption("logo",0) = "<c>This is our logo<br><c> <img>logo</img>"
.EndUpdate
End With
```

### VB6

With Gauge1

```
.BeginUpdate
.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
```

```

1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.Caption(exLayerCaption) = "This is just a caption"
.ExtraCaption("logo",exLayerCaptionAnchor) = 2
.ExtraCaption("logo",exLayerCaptionWordWrap) = True
.ExtraCaption("logo",exLayerCaptionWidth) = "164"
.ExtraCaption("logo",exLayerCaptionLeft) = "width - 164"
.ExtraCaption("logo",exLayerCaption) = "<c>This is our logo<br> <c>
<img>logo</img>"
.EndUpdate
End With

```

## VB.NET

```

With Exgauge1
.BeginUpdate()
.set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5

.set_Caption(exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"This
is just a caption")

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap

.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
- 164")

```

```
.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCap
<c>This is our logo<br><c><img>logo</img>")
.EndUpdate()
End With
```

## VB.NET for /COM

```
With AxGauge1
.BeginUpdate()
.set_HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"This is just
a caption")

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnch

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWor

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidt

.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft,'
- 164")
.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"
<c>This is our logo<br><c><img>logo</img>")
.EndUpdate()
End With
```

## C++

```
/*
Copy and paste the following directives to your header file as
```

*it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control Library'*

```
#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1-
> PutHTMLPicture(L"logo", "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
spGauge1-> PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
spGauge1-> PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1-> GetLayers()-> PutCount(5);
spGauge1-> PutCaption(EXGAUGELib::exLayerCaption, "This is just a caption");
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionAnchor, long(2));
spGauge1-
> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWordWrap, VARIANT_TRUE);
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionWidth, "164");
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaptionLeft, "width - 164");
spGauge1-> PutExtraCaption("logo", EXGAUGELib::exLayerCaption, "<c>This is our
logo<br> <c> <img>logo</img>");
spGauge1-> EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();
Gauge1->HTMLPicture[L"logo"] =
TVariant("E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png";
Gauge1->Layers->Count = 5;
Gauge1-> Caption[Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] =
TVariant("This is just a caption");
```

Gauge1-

```
> ExtraCaption[TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] = TVariant(2);
```

Gauge1-

```
> ExtraCaption[TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] = TVariant(true);
```

Gauge1-

```
> ExtraCaption[TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] = TVariant("164");
```

Gauge1-

```
> ExtraCaption[TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] = TVariant("width - 164");
```

Gauge1-

```
> ExtraCaption[TVariant("logo"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] = TVariant("<c> This is our logo<br> <c> <img> logo</img> ");
```

```
Gauge1->EndUpdate();
```

## C#

```
exgauge1.BeginUpdate();
```

```
exgauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
```

```
exgauge1.PicturesPath = "C:\\Program Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
```

```
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
```

```
exgauge1.Layers.Count = 5;
```

```
exgauge1.set_Caption(exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionCaption, "is just a caption");
```

```
exgauge1.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionCaption, "is just a caption");
```

```
exgauge1.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionCaption, "is just a caption");
```

```
exgauge1.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionCaption, "is just a caption");
```

```
exgauge1.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionCaption, "is just a caption");
```

```
exgauge1.set_ExtraCaption("logo",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionCaption, "is just a caption");
```

```
<c>This is our logo<br><c><img>logo</img>");  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"></OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.BeginUpdate();  
    Gauge1.HTMLPicture("logo") = "E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png";  
    Gauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 5;  
    Gauge1.Caption(0) = "This is just a caption";  
    Gauge1.ExtraCaption("logo",3) = 2;  
    Gauge1.ExtraCaption("logo",8) = true;  
    Gauge1.ExtraCaption("logo",6) = "164";  
    Gauge1.ExtraCaption("logo",4) = "width - 164";  
    Gauge1.ExtraCaption("logo",0) = "<c>This is our logo<br><c>  
<img>logo</img>";  
    Gauge1.EndUpdate();  
}  
</SCRIPT>  
</BODY>
```

## VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"></OBJECT>
```

```

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Gauge1
    .BeginUpdate
    .HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 5
    .Caption(0) = "This is just a caption"
    .ExtraCaption("logo",3) = 2
    .ExtraCaption("logo",8) = True
    .ExtraCaption("logo",6) = "164"
    .ExtraCaption("logo",4) = "width - 164"
    .ExtraCaption("logo",0) = "<c>This is our logo<br><c><img>logo</img>"
    .EndUpdate
  End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.set_HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 5;
axGauge1.set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,"This
is just a caption");
axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa

axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa

axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa

```



```

axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
- 164");
axGauge1.set_ExtraCaption("logo",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
<c>This is our logo<br> <c> <img>logo</img>");
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    ;

    super();

    exgauge1.BeginUpdate();
    exgauge1.HTMLPicture("logo","E:\\Exontrol\\Exontrol.Logo\\exontrol.logo.png");
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(5);
    exgauge1.Caption(0/*exLayerCaption*/, "This is just a caption");

    exgauge1.ExtraCaption("logo",3/*exLayerCaptionAnchor*/,COMVariant::createFromInt(

    exgauge1.ExtraCaption("logo",8/*exLayerCaptionWordWrap*/,COMVariant::createFrom

    exgauge1.ExtraCaption("logo",6/*exLayerCaptionWidth*/,"164");
    exgauge1.ExtraCaption("logo",4/*exLayerCaptionLeft*/,"width - 164");
    exgauge1.ExtraCaption("logo",0/*exLayerCaption*/,"<c>This is our logo<br> <c>
<img>logo</img>");
    exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
  BeginUpdate();
  set_HTMLPicture('logo','E:\Exontrol\Exontrol.Logo\exontrol.logo.png');
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,'This is just a
caption');

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionAnch

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWord'

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionWidth

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft,'w
- 164');

  set_ExtraCaption('logo',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,'<c>T
is our logo<br><c> <img>logo</img>');
  EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
  BeginUpdate();
  HTMLPicture['logo'] := 'E:\Exontrol\Exontrol.Logo\exontrol.logo.png';
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
  PicturesName := 'Layer` + int(value + 1) + `.png`;

```

```

Layers.Count := 5;
Caption[EXGAUGELib_TLB.exLayerCaption] := 'This is just a caption';
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionAnchor] := OleVariant(2);
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWordWrap] :=
OleVariant(True);
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionWidth] := '164';
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaptionLeft] := 'width - 164';
ExtraCaption['logo',EXGAUGELib_TLB.exLayerCaption] := '<c>This is our logo<br>
<c> <img>logo</img>';
EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.Object.HTMLPicture("logo") = "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
.Object.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.Object.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Object.Layers.Count = 5
.Object.Caption(0) = "This is just a caption"
.Object.ExtraCaption("logo",3) = 2
.Object.ExtraCaption("logo",8) = .T.
.Object.ExtraCaption("logo",6) = "164"
.Object.ExtraCaption("logo",4) = "width - 164"
.Object.ExtraCaption("logo",0) = "<c>This is our logo<br> <c>
<img>logo</img>"
.Object.EndUpdate
endwith

```

## dBASE Plus

local oGauge

```

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.Template = [HTMLPicture("logo") =

```

```

"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"] // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.Template = [Caption(0) = "This is just a caption"] // oGauge.Caption(0) =
"This is just a caption"
oGauge.Template = [ExtraCaption("logo",3) = 2] // oGauge.ExtraCaption("logo",3) = 2
oGauge.Template = [ExtraCaption("logo",8) = True] // oGauge.ExtraCaption("logo",8)
= true
oGauge.Template = [ExtraCaption("logo",6) = "164"] // oGauge.ExtraCaption("logo",6)
= "164"
oGauge.Template = [ExtraCaption("logo",4) = "width - 164"] //
oGauge.ExtraCaption("logo",4) = "width - 164"
oGauge.Template = [ExtraCaption("logo",0) = "<c>This is our logo<br><c>
<img>logo</img>"] // oGauge.ExtraCaption("logo",0) = "<c>This is our logo<br>
<c><img>logo</img>"
oGauge.EndUpdate()

```

## XBasic (Alpha Five)

```

Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.Template = "HTMLPicture(`logo`) =
`E:\Exontrol\Exontrol.Logo\exontrol.logo.png`" // oGauge.HTMLPicture("logo") =
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.Template = "Caption(0) = `This is just a caption`" // oGauge.Caption(0) =
"This is just a caption"
oGauge.Template = "ExtraCaption(`logo`,3) = 2" // oGauge.ExtraCaption("logo",3) = 2

```

```

oGauge.Template = "ExtraCaption('logo`,8) = True" // oGauge.ExtraCaption("logo",8)
= .t.
oGauge.Template = "ExtraCaption('logo`,6) = `164`" // oGauge.ExtraCaption("logo",6)
= "164"
oGauge.Template = "ExtraCaption('logo`,4) = `width - 164`" //
oGauge.ExtraCaption("logo",4) = "width - 164"
oGauge.Template = "ExtraCaption('logo`,0) = `

```

## Visual Objects

```

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:[HTMLPicture,"logo"] :=
"E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 5
oDCOCX_Exontrol1:[Caption,exLayerCaption] := "This is just a caption"
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionAnchor] := 2
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionWordWrap] := true
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionWidth] := "164"
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaptionLeft] := "width - 164"
oDCOCX_Exontrol1:[ExtraCaption,"logo",exLayerCaption] := "<c>This is our
logo<br><c><img>logo</img>"
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge

oGauge = ole_1.Object
oGauge.BeginUpdate()

```

```

oGauge.HTMLPicture("logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.Caption(0,"This is just a caption")
oGauge.ExtraCaption("logo",3,2)
oGauge.ExtraCaption("logo",8,true)
oGauge.ExtraCaption("logo",6,"164")
oGauge.ExtraCaption("logo",4,"width - 164")
oGauge.ExtraCaption("logo",0,"<c>This is our logo<br><c><img>logo</img>")
oGauge.EndUpdate()

```

## Visual DataFlex

```

Procedure OnCreate
    Forward Send OnCreate
    Send ComBeginUpdate
    Set ComHTMLPicture "logo" to "E:\Exontrol\Exontrol.Logo\exontrol.logo.png"
    Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
    Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
    Variant voLayers
    Get ComLayers to voLayers
    Handle hoLayers
    Get Create (RefClass(cComLayers)) to hoLayers
    Set pvComObject of hoLayers to voLayers
        Set ComCount of hoLayers to 5
    Send Destroy to hoLayers
    Set ComCaption OLEexLayerCaption to "This is just a caption"
    Set ComExtraCaption "logo" OLEexLayerCaptionAnchor to 2
    Set ComExtraCaption "logo" OLEexLayerCaptionWordWrap to True
    Set ComExtraCaption "logo" OLEexLayerCaptionWidth to "164"
    Set ComExtraCaption "logo" OLEexLayerCaptionLeft to "width - 164"
    Set ComExtraCaption "logo" OLEexLayerCaption to "<c>This is our logo<br><c>
<img>logo</img>"

```

```
Send ComEndUpdate
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480} ,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()

    oGauge:SetProperty("HTMLPicture","logo","E:\Exontrol\Exontrol.Logo\exontrol.logo.png")

    oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 5
    oGauge:SetProperty("Caption",0/*exLayerCaption*/,"This is just a caption")
    oGauge:SetProperty("ExtraCaption","logo",3/*exLayerCaptionAnchor*/2)
    oGauge:SetProperty("ExtraCaption","logo",8/*exLayerCaptionWordWrap*/,.T.)
    oGauge:SetProperty("ExtraCaption","logo",6/*exLayerCaptionWidth*/,"164")
    oGauge:SetProperty("ExtraCaption","logo",4/*exLayerCaptionLeft*/,"width -
164")
```

```
oGauge:SetProperty("ExtraCaption","logo",0/*exLayerCaption*/,"<c>This is our  
logo<br><c><img>logo</img>" )
```

```
oGauge:EndUpdate()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```



# property Gauge.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object used to paint the items.

Use the Font property to change the control's font . Use the [Refresh](#) method to refresh the control. Use the [BeginUpdate](#) and [EndUpdate](#) method to maintain performance while adding new layers to the control.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows an extra-caption associated with the layer:



# property Gauge.ForeColor as Color

Specifies the control's foreground color.

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

The ForeColor property specifies the control's foreground color. The [BackColor](#) property specifies the control's background color.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.

The following screen shot shows an extra-caption associated with the layer:



# method Gauge.FormatABC (Expression as String, [A as Variant], [B as Variant], [C as Variant])

Formats the A,B,C values based on the giving expression and returns the result.

Type	Description
Expression as String	A String that defines the expression to be evaluated.
A as Variant	A VARIANT expression that indicates the value of the A keyword.
B as Variant	A VARIANT expression that indicates the value of the B keyword.
C as Variant	A VARIANT expression that indicates the value of the C keyword.

Return	Description
Variant	A VARIANT expression that indicates the result of the evaluation the Expression.

The FormatABC method formats the A,B,C values based on the giving expression and returns the result. 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.

For instance:

- "A + B + C", adds / concatenates the values of the A, B and C
- "value MIN 0 MAX 99", limits the value between 0 and 99
- "value format ``,", formats the value with two decimals, according to the control's panel setting
- "date(`now`)" returns the current time as double

The Expression of the FormatABC method supports the following keywords, constants, operators and functions:

- **A** or **value** keyword, indicates a variable A whose value is giving by the A parameter
- **B** keyword, indicates a variable B whose value is giving by the B parameter
- **C** keyword, indicates a variable C whose value is giving by the C parameter

*The constants are ( DPI-Aware components ):*

- **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 y-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** ( remainder 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** ( and 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 :*

- **MIN** ( 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.

- **MAX** ( 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 than *if* (immediate if operator) alternative.

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

***expression case ([default : default\_expression ; ] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)***

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

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

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

- 0 - empty ( not initialized )
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency

- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"
- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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



# property Gauge.FormatAnchor(New as Boolean) as String

Specifies the visual effect for anchor elements in HTML captions.

Type	Description
New as Boolean	A Boolean expression that indicates whether to specify the anchors never clicked or anchors being clicked.
String	A String expression that indicates the HTMLformat to apply to anchor elements.

By default, the FormatAnchor(**True**) property is "<u><fgcolor=0000FF>#" that indicates that the anchor elements ( that were never clicked ) are underlined and shown in light blue. Also, the FormatAnchor(**False**) property is "<u><fgcolor=000080>#" that indicates that the anchor elements are underlined and shown in dark blue. The visual effect is applied to the anchor elements, if the FormatAnchor property is not empty. For instance, if you want to do not show with a new effect the clicked anchor elements, you can use the FormatAnchor(**False**) = "", that means that the clicked or not-clicked anchors are shown with the same effect that's specified by FormatAnchor(**True**). An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the [AnchorClick](#) event to notify that the user clicks an anchor element. This event is fired only if prior clicking the control it shows the hand cursor. The AnchorClick event carries the identifier of the anchor, as well as application options that you can specify in the anchor element. The hand cursor is shown when the user hovers the mouse on the anchor elements.

# method Gauge.FreezeEvents (Freeze as Boolean)

Prevents the control to fire any event.

Type	Description
Freeze as Boolean	A Boolean expression that specifies whether the control' events are froze or unfroze

The FreezeEvents(True) method freezes the control's events until the FreezeEvents(False) method is called. You can use the FreezeEvents method to enable / disable the control's events.

# property Gauge.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none"><li>• a string expression that indicates the path to the picture file, being loaded.</li><li>• a string expression that indicates the base64 encoded string that holds a picture object, Use the Exontrol's <a href="#">ExImages</a> Tool to save your picture as base64 encoded format.</li><li>• A Picture object that indicates the picture being added or replaced. ( A Picture object implements IPicture interface ),</li></ul> <p>If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added.</p>

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

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

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

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





# property Gauge.hWnd as Long

Retrieves the control's window handle.

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

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

# method Gauge.Images (Handle as Variant)

Sets a runtime the control's image tree.

Type	Description
Handle as Variant	<p>The Handle parameter can be:</p> <ul style="list-style-type: none"><li>• 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 (<i>string, loads the icon using its path</i>)</li><li>• A string expression that indicates the BASE64 encoded string that holds the icons list. Use the Exontrol's <a href="#">ExImages</a> tool to save/load your icons as BASE64 encoded format. In this case the string may begin with "gBJJ..." (<i>string, loads icons using base64 encoded string</i>)</li><li>• A reference to a Microsoft ImageList control (mscomctl.ocx, MSComctlLib.ImageList type) that holds the icons to add (<i>object, loads icons from a Microsoft ImageList control</i>)</li><li>• 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 (<i>object, loads icon from a Picture object</i>)</li><li>• A long expression that identifies a handle to an Image List Control ( the Handle should be of HIMAGELIST type ). On 64-bit platforms, the Handle parameter must be a Variant of LongLong / LONG_PTR data type ( signed 64-bit (8-byte) integers ), saved under lVal field, as VT_I8 type. The LONGLONG / LONG_PTR is __int64, a 64-bit integer. For instance, in C++ you can use as Images( COleVariant( LONG_PTR)hImageList) ) or Images( COleVariant( LONGLONG)hImageList) ), where hImageList is of</li></ul>

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

---

The user can add images at design time, by drag and drop files to combo's image holder. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. Use the [Replacelcon](#) method to add, remove or clear icons in the control's images collection.

# property Gauge.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 Gauge.LayerAutoSize as Long

Specifies the index of the layer that determines the size to display all layers.

Type	Description
Long	A Long expression that indicates the index of the layer that determines the size to display all layers

By default, the LayerAutoSize property is 0, which indicates that the size of the entire view is defined by the size of the first layer. The size of the layer is determined by it's picture ( [Picture](#) property ). Shortly, the LayerAutoSize resizes all layers based on the picture of the first layer.

For instance, you can use the LayerAutoSize property to:

- stretches all the layers to the control's view, if the LayerAutoSize property is -1 ( or any other value that's not an index in the Layers collection )
- resizes all layers relative to a specified layer and it's background picture
- resizes all layers to a specified layer, whose [Width](#) and [Height](#) properties specify the size of the view.

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move /resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following samples show how you can resize the control's view to 164 x 164 pixels, by adding a new hidden layer called "autosize":

### VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 10
    With .Layers.Add("autosize")
        .Visible = False
        .Width = 164
        .Height = 164
    End With
    .LayerAutoSize = .Layers.Item("autosize").Index
    .EndUpdate
End With
```

### VB6

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
```

```

.Layers.Count = 10
With .Layers.Add("autosize")
    .Visible = False
    .Width = 164
    .Height = 164
End With
.LayerAutoSize = .Layers.Item("autosize").Index
.EndUpdate
End With

```

## VB.NET

```

With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 10
    With .Layers.Add("autosize")
        .Visible = False
        .Width = 164
        .Height = 164
    End With
    .LayerAutoSize = .Layers.Item("autosize").Index
    .EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 10
    With .Layers.Add("autosize")
        .Visible = False
        .Width = 164

```

```
.Height = 164
End With
.LayerAutoSize = .Layers.Item("autosize").Index
.EndUpdate()
End With
```

## C++

```
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
    Library'

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
>GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(10);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->Add("autosize");
var_Layer->PutVisible(VARIANT_FALSE);
var_Layer->PutWidth(L"164");
var_Layer->PutHeight(L"164");
spGauge1->PutLayerAutoSize(spGauge1->GetLayers()->GetItem("autosize")-
>GetIndex());
spGauge1->EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`";
```



```

Gauge1->Layers->Count = 10;
ExgaugeLib_tlb::ILayerPtr var_Layer = Gauge1->Layers->Add(TVariant("autosize"));
    var_Layer->Visible = false;
    var_Layer->Width = L"164";
    var_Layer->Height = L"164";
Gauge1->LayerAutoSize = Gauge1->Layers->get_Item(TVariant("autosize"))-
>Index;
Gauge1->EndUpdate();

```

## C#

```

exgauge1.BeginUpdate();
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
exgauge1.PicturesName = "\\Layer` + int(value + 1) + `.png`;
exgauge1.Layers.Count = 10;
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers.Add("autosize");
    var_Layer.Visible = false;
    var_Layer.Width = 164.ToString();
    var_Layer.Height = 164.ToString();
exgauge1.LayerAutoSize = exgauge1.Layers["autosize"].Index;
exgauge1.EndUpdate();

```

## JScript/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "\\Layer` + int(value + 1) + `.png`;

```

```

Gauge1.Layers.Count = 10;
var var_Layer = Gauge1.Layers.Add("autosize");
    var_Layer.Visible = false;
    var_Layer.Width = 164;
    var_Layer.Height = 164;
Gauge1.LayerAutoSize = Gauge1.Layers.Item("autosize").Index;
Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = "`Layer` + int(value + 1) + `.png`"
        .Layers.Count = 10
        With .Layers.Add("autosize")
            .Visible = False
            .Width = 164
            .Height = 164
        End With
        .LayerAutoSize = .Layers.Item("autosize").Index
        .EndUpdate
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```
axGauge1.BeginUpdate();
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 10;
EXGAUGELib.Layer var_Layer = axGauge1.Layers.Add("autosize");
    var_Layer.Visible = false;
    var_Layer.Width = 164.ToString();
    var_Layer.Height = 164.ToString();
axGauge1.LayerAutoSize = axGauge1.Layers["autosize"].Index;
axGauge1.EndUpdate();
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Layer;
    anytype var_Layer;
    ;

    super();

    exgauge1.BeginUpdate();
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(10);
    var_Layer = COM::createFromObject(exgauge1.Layers()).Add("autosize"); com_Layer
= var_Layer;
    com_Layer.Visible(false);
    com_Layer.Width(164);
    com_Layer.Height(164);
    exgauge1.LayerAutoSize(exgauge1.Layers().Item("autosize").Index());
```

```
exgauge1.EndUpdate();  
}
```

## Delphi 8 (.NET only)

```
with AxGauge1 do  
begin  
  BeginUpdate();  
  PicturesPath := 'C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';  
  PicturesName := 'Layer` + int(value + 1) + `.png`';  
  Layers.Count := 10;  
  with Layers.Add('autosize') do  
  begin  
    Visible := False;  
    Width := 164;  
    Height := 164;  
  end;  
  LayerAutoSize := Layers.Item['autosize'].Index;  
  EndUpdate();  
end
```

## Delphi (standard)

```
with Gauge1 do  
begin  
  BeginUpdate();  
  PicturesPath := 'C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';  
  PicturesName := 'Layer` + int(value + 1) + `.png`';  
  Layers.Count := 10;  
  with Layers.Add('autosize') do  
  begin  
    Visible := False;  
    Width := 164;  
    Height := 164;  
  end;  
  LayerAutoSize := Layers.Item['autosize'].Index;
```

```
EndUpdate();  
end
```

## VFP

```
with thisform.Gauge1  
  .BeginUpdate  
  .PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
  .PicturesName = "`Layer` + int(value + 1) + `.png`"  
  .Layers.Count = 10  
  with .Layers.Add("autosize")  
    .Visible = .F.  
    .Width = 164  
    .Height = 164  
  endwith  
  .LayerAutoSize = .Layers.Item("autosize").Index  
  .EndUpdate  
endwith
```

## dBASE Plus

```
local oGauge,var_Layer  
  
oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 10  
var_Layer = oGauge.Layers.Add("autosize")  
  var_Layer.Visible = false  
  var_Layer.Width = Str(164)  
  var_Layer.Height = Str(164)  
oGauge.LayerAutoSize = oGauge.Layers.Item("autosize").Index  
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P
Dim var_Layer as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 10
var_Layer = oGauge.Layers.Add("autosize")
    var_Layer.Visible = .f
    var_Layer.Width = 164
    var_Layer.Height = 164
oGauge.LayerAutoSize = oGauge.Layers.Item("autosize").Index
oGauge.EndUpdate()
```

## Visual Objects

```
local var_Layer as ILayer

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 10
var_Layer := oDCOCX_Exontrol1:Layers:Add("autosize")
    var_Layer:Visible := false
    var_Layer:Width := AsString(164)
    var_Layer:Height := AsString(164)
oDCOCX_Exontrol1:LayerAutoSize := oDCOCX_Exontrol1:Layers:
[Item,"autosize"]:Index
oDCOCX_Exontrol1:EndUpdate()
```

## PowerBuilder

OleObject oGauge,var\_Layer

oGauge = ole\_1.Object

oGauge.BeginUpdate()

oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"

oGauge.Layers.Count = 10

var\_Layer = oGauge.Layers.Add("autosize")

var\_Layer.Visible = false

var\_Layer.**Width** = String(164)

var\_Layer.**Height** = String(164)

oGauge.**LayerAutoSize** = oGauge.Layers.Item("autosize").Index

oGauge.EndUpdate()

## Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Send ComBeginUpdate

Set ComPicturesPath to "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Set ComCount of hoLayers to 10

Send Destroy to hoLayers

Variant voLayers1

Get ComLayers to voLayers1

Handle hoLayers1

Get Create (RefClass(cComLayers)) to hoLayers1

Set pvComObject of hoLayers1 to voLayers1

Variant voLayer

```

Get ComAdd of hoLayers1 "autosize" to voLayer
Handle hoLayer
Get Create (RefClass(cComLayer)) to hoLayer
Set pvComObject of hoLayer to voLayer
    Set ComVisible of hoLayer to False
    Set ComWidth of hoLayer to 164
    Set ComHeight of hoLayer to 164
Send Destroy to hoLayer
Send Destroy to hoLayers1
Variant v
Variant voLayers2
Get ComLayers to voLayers2
Handle hoLayers2
Get Create (RefClass(cComLayers)) to hoLayers2
Set pvComObject of hoLayers2 to voLayers2
    Variant voLayer1
    Get ComItem of hoLayers2 "autosize" to voLayer1
    Handle hoLayer1
    Get Create (RefClass(cComLayer)) to hoLayer1
    Set pvComObject of hoLayer1 to voLayer1
        Get ComIndex of hoLayer1 to v
    Send Destroy to hoLayer1
Send Destroy to hoLayers2
Set ComLayerAutoSize to v
Send ComEndUpdate
End_Procedure

```

## XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge
    LOCAL oLayer

```



```

oForm := XbpDialog():new( AppDesktop() )
oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

oGauge:BeginUpdate()
oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
oGauge:Layers():Count := 10
oLayer := oGauge:Layers():Add("autosize")
oLayer:Visible := .F.
oLayer:Width := Transform(164,"")
oLayer:Height := Transform(164,"")
oGauge:LayerAutoSize := oGauge:Layers:Item("autosize"):Index()
oGauge:EndUpdate()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

# property Gauge.LayerClipTo as Long

Specifies the index of the layer that clips the entire control to.

Type	Description
Long	A Long expression that specifies the index of the layer that clips the entire control to.

By default, the LayerClipTo property is -1, which indicates that it has no effect. The LayerClipTo property specifies the index of the layer that clips the entire control to. The [LayerClipToAlpha](#) property returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region. The [LayerClipToParent](#) property indicates if the LayerClipTo method clips the control itself, parent or the owner of the control. The [AllowMoveOnClick](#) property allows moving the window that contains the control to a new position, as you would do by clicking the form's title/caption. The [LayerUpdate](#) property helps you to create a smooth widget on the screen or form.

"I would like to put the control on a form, then make the form transparent so the control appears on the desktop with just the images contained in the layers visible. For example, take a clock example and make the control background and the form transparent, and you have a working clock widget."

The control support transparent form, or in other words, displaying the control's itself without its form behind. In order to make your eXGauge control to display a widget, ( no form behind or form transparent ), you need to use the following properties:

- **LayerClipTo** property of the control, specifies the index of the layer that clips the entire control to. By default, the LayerClipTo property is -1, which indicates that no clipping is supported. So, one of the layers that composes your widget must be specified as the widget's background, and so, the entire view of the control is clipped to region defined by the clipping layer (LayerClipTo). The LayerClipTo property may refer to any layer, visible or hidden, which includes a picture or a clipping object ( Clip property ).
- [Layer.LayerClipToAlpha](#) property of the Layer object, returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region. By default, the LayerClipToAlpha property is 0, which indicates that only pixels of the layer that has 0 on the alpha channel (transparent pixels) defines the transparent region, and so the clipping region. In other words, the value from 0 to LayerClipToAlpha defines transparent pixels, and the rest defines the opaque pixels to be included in the clipping region. So based on the layer's picture, you can change the LayerClipToAlpha property for a better look of your widget.
- [LayerClipToParent](#) property of the control, indicates if the LayerClipTo method clips the control itself, parent or the owner of the control. By default, the LayerClipToParent

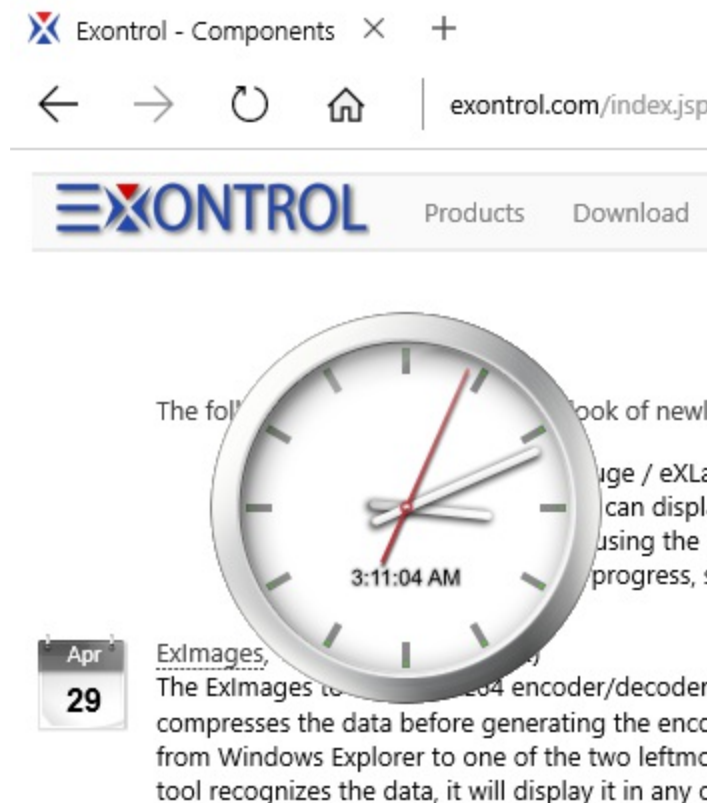
property is `exLayerUpdateControl`, which indicates that the control's itself is clipped relative to its form that hosts it. Change the `LayerClipToParent` property to `exLayerUpdateScreen`, or `exLayerUpdateParent`, and so the clipping region is applied to its form/dialog/parent window.

The following VB sample defines the control as a widget:

```
With Gauge1
    .LayerClipTo = 0
    .LayerClipToParent = exLayerUpdateScreen
End With
```

The sample defines the layer with the Index 0, as being the clipping layer. The setup installs the `C:\Program Files\Exontrol\ExGauge\Sample\VB\Clock-Widget-Region` that shows all these working.

The following screen shot shows the control on a transparent form:



The following screen shot shows the control on an opaque form:

**EXONTROL**

Products

Download

http://www.e... - □ X



newl

eXL2

displ

the

ess, !

oder

encc

eftmc

tool recognizes the data, it will display it in any c

# property Gauge.LayerClipToParent as LayerUpdateEnum

Indicates if the LayerClipTo method clips the control itself, parent or the owner of the control.

Type	Description
<a href="#">LayerUpdateEnum</a>	A LayerUpdateEnum expression that specifies the

By default, the LayerClipToParent property is exLayerUpdateControl, which indicates that the control's itself is clipped relative to its form that hosts it. Change the LayerClipToParent property to exLayerUpdateScreen, or exLayerUpdateParent, and so the clipping region is applied to its form/dialog/parent window. The [LayerClipTo](#) property specifies the index of the layer that clips the entire control to. The [LayerClipToAlpha](#) property returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region.

*"I would like to put the control on a form, then make the form transparent so the control appears on the desktop with just the images contained in the layers visible. For example, take a clock example and make the control background and the form transparent, and you have a working clock widget."*

The control support transparent form, or in other words, displaying the control's itself without its form behind. In order to make your eXGauge control to display a widget, ( no form behind or form transparent ), you need to use the following properties:

- **LayerClipTo** property of the control, specifies the index of the layer that clips the entire control to. By default, the LayerClipTo property is -1, which indicates that no clipping is supported. So, one of the layers that composes your widget must be specified as the widget's background, and so, the entire view of the control is clipped to region defined by the clipping layer (LayerClipTo). The LayerClipTo property may refer to any layer, visible or hidden, which includes a picture or a clipping object ( Clip property ).
- [Layer.LayerClipToAlpha](#) property of the Layer object, returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region. By default, the LayerClipToAlpha property is 0, which indicates that only pixels of the layer that has 0 on the alpha channel (transparent pixels) defines the transparent region, and so the clipping region. In other words, the value from 0 to LayerClipToAlpha defines transparent pixels, and the rest defines the opaque pixels to be included in the clipping region. So based on the layer's picture, you can change the LayerClipToAlpha property for a better look of your widget.
- [LayerClipToParent](#) property of the control, indicates if the LayerClipTo method clips the control itself, parent or the owner of the control. By default, the LayerClipToParent property is exLayerUpdateControl, which indicates that the control's itself is clipped relative to its form that hosts it. Change the LayerClipToParent property to

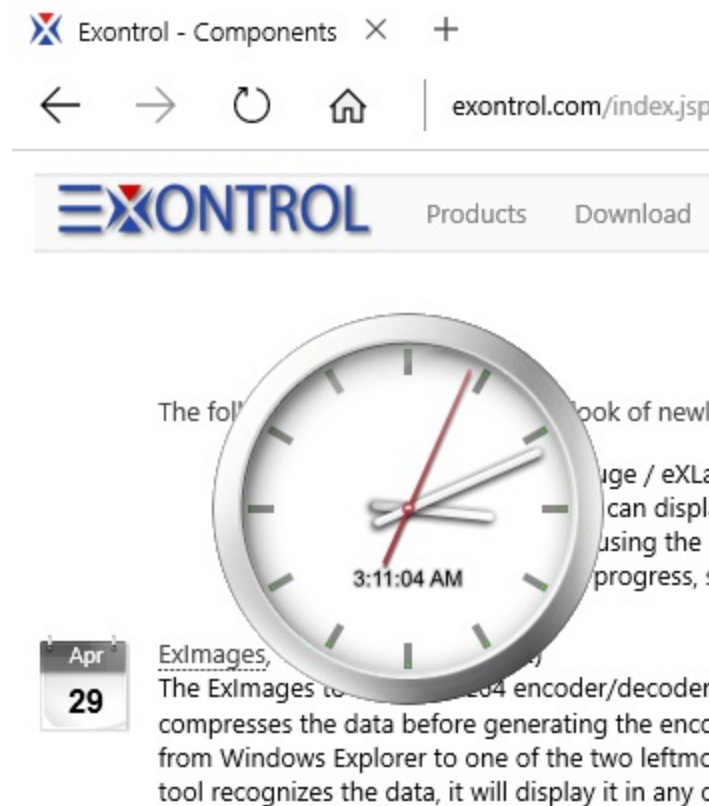
exLayerUpdateScreen, or exLayerUpdateParent, and so the clipping region is applied to its form/dialog/parent window.

For instance, the following VB sample defines the control as a widget:

```
With Gauge1
    .LayerClipTo = 0
    .LayerClipToParent = exLayerUpdateScreen
End With
```

The sample defines the layer with the Index 0, as being the clipping layer. The setup installs the C:\Program Files\Exontrol\ExGauge\Sample\VB\Clock-Widget-Region that shows all these working.

The following screen shot shows the control on a transparent form:



The following screen shot shows the control on an opaque form:

**EXONTROL**

Products

Download

http://www.e... - □ ×



newl

eXL2

displ

the

ess, !

oder

encc

eftmc

tool recognizes the data, it will display it in any c

# property Gauge.LayerDragAny as Long

Specifies the index of the layer to drag (rotate or move) once the user clicks anywhere on the control.

Type	Description
Long	A long expression that specifies the index of the layer to drag (rotate or move) once the user clicks anywhere on the control.

By default, the LayerDragAny property is -1, which indicates that has no effect. The LayerDragAny property specifies the index of the layer to drag (rotate or move) once the user clicks anywhere on the control. For instance, if the LayerDragAny property is 0, it means that the layer with the index-0 is always dragging no matter where the cursor is.



# property Gauge.LayerFromPoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS) as Long

Retrieves the index of the layer from the point ( only visible and selectable objects are included ).

Type	Description
X as OLE_XPOS_PIXELS	A long expression that specifies the x-position in client coordinate to get the layer from.
Y as OLE_YPOS_PIXELS	A long expression that specifies the y-position in client coordinate to get the layer from.
Long	A Long expression that specifies the index of the layer from the cursor, or -1 if not found.

The LayerFromPoint property retrieves the layer from point that's visible and selectable. The [Visible](#) property shows or hides a specific layer (visible). The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [Item](#) property accesses the Layer object giving its index. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). Use the [ShowToolTip](#) method to display a custom tooltip. The [ToolTip](#) / [ToolTipTitle](#) property indicates the layer's tooltip. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects.

# property Gauge.LayerOfValue as Long

Specifies the index of the layer whose value represents the control's Value property.

Type	Description
Long	A Long expression that specifies the index of the layer whose <a href="#">Value</a> represents the control's <a href="#">Value</a> property, or -1 to which indicates that last visible layer with <a href="#">OnDrag</a> property set.

By default, the LayerOfValue property is -1, which indicates that the last visible layer whose [OnDrag](#) property is not exDoNothing, is the layer that specifies the control's value. The LayerOfValue property specifies the index of the layer whose value represents the control's Value property. The layer's [Value](#) could indicate its offset or its rotation angle, based on the [OnDrag](#) property. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). Use the [Value](#) property of the Clip object to associate a value with the layer's clipping region. Each layer can associate a value with it, while the control's Value property can be associated through the LayerOfValue property with the value of one of the layers within the control.

For instance:

- the control displays a clock, the value could be the current-time
- the control shows a switch, so the value could indicate the state of the switch
- the control shows a thermometer, so the value could be the current temperature
- the control displays a gauge, so the value could be the value on the gauge pointed by the needle

The [Change](#) event occurs when the layer's Value property is changed. During the Change event, you can change values of other layers as well. For instance, if the second-hand of the clock is rotated, you can rotate the hour and the minute-hands of the clock as well. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. *You can call DragInfo.Debug = -1 during the [DragStart](#) event to display debugging information like current movement, rotation angle when drag operation is performed.*

The Value property indicates the **value** keyword in the following properties:

- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the [ValueToOffsetX](#) expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a

value.

- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The Value property works in association with the layer's [OnDrag](#) property like follows:

- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetX](#) property indicates the layer's [OffsetX](#) property.
- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetY](#) property indicates the layer's [OffsetY](#) property.
- If the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**, evaluating the [ValueToRotateAngle](#) property indicates the layer's [RotateAngle](#) property.

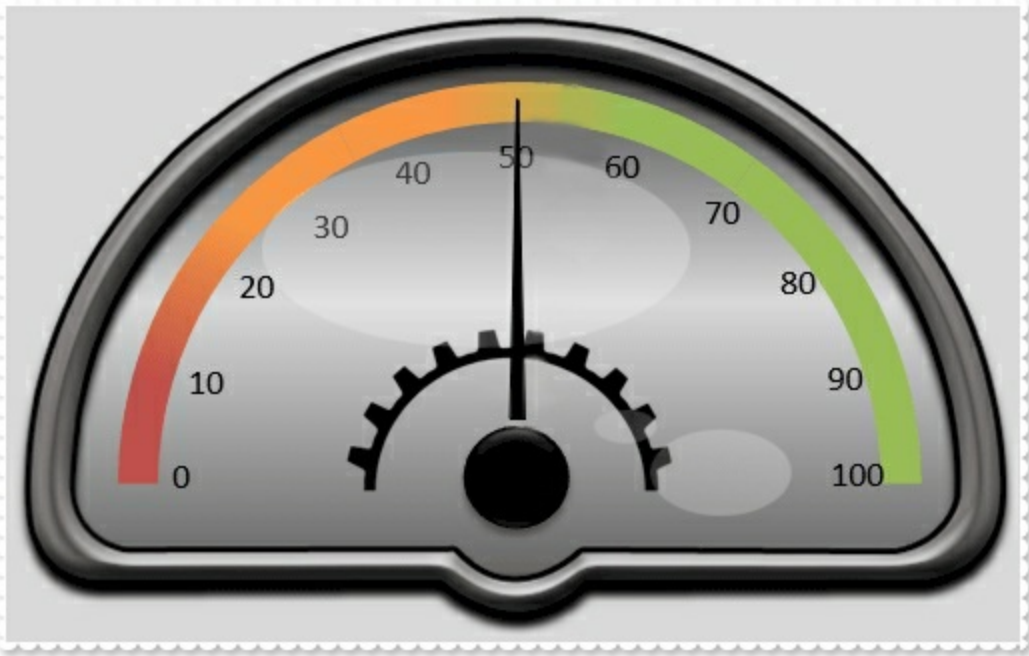
# property Gauge.Layers as Layers

Returns the Layers collection.

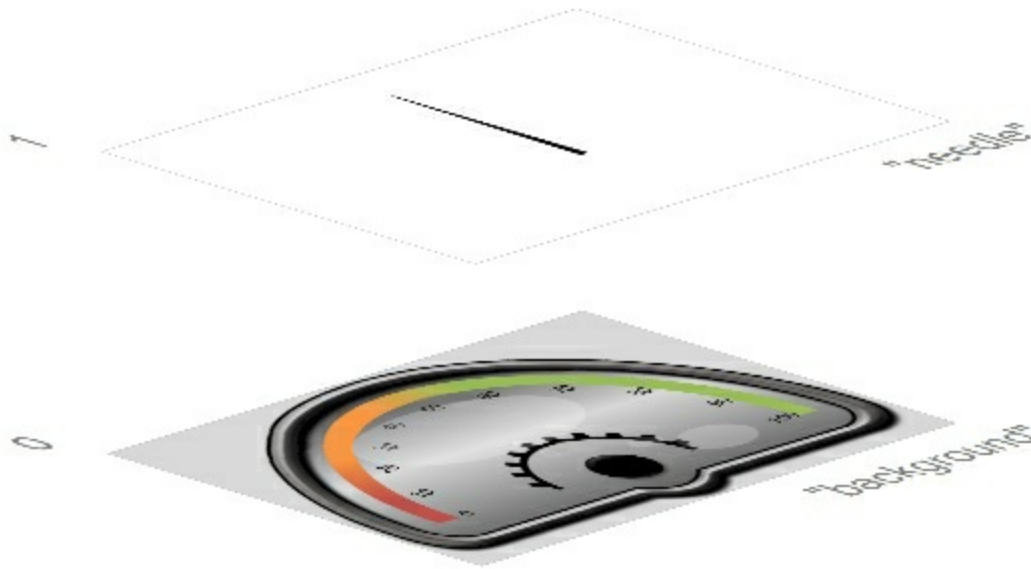
Type	Description
<a href="#">Layers</a>	A <a href="#">Layers</a> collection of <a href="#">Layer</a> objects.

The Layers property gives access to the control's Layers collection. Any layer can display unlimited opaque / transparent graphics, HTML text, can be visible, selectable, draggable and so on. Any layer can change its position in the layers collection as well. The Layer can change its brightness, contrast, grayscale or transparency as well.

The following screen shot shows a control composed by two pictures:  
"Guage\_Background.png" and "Guage\_Needle.png" from the "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Guage" folder



The following picture shows how layers "background" and "needle" composes the control's view



The following properties can be used to add / remove layers within the control:

- [Count](#) property, adds / removes layers to / from the control
- [Add](#) method, adds a new layer to the control.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

# property Gauge.LayerUpdate as LayerUpdateEnum

Specifies where the control updates its content.

Type	Description
<a href="#">LayerUpdateEnum</a>	A LayerUpdateEnum expression that specifies where the control updates its content.

By default, the LayerUpdate property property is exLayerUpdateControl, which indicates that the control's content is shown on the control itself ( no effect ). The LayerUpdate property indicates where the control's content is updated. The control support transparent form, or in other words, displaying the control's itself without its form behind. The [AllowMoveOnClick](#) property allows moving the window that contains the control to a new position, as you would do by clicking the form's title/caption. The [LayerClipTo](#) property specifies the index of the layer that clips the entire control to.

In order to make your eXGauge control to display a widget, ( no form behind or form transparent ), you need to use the following properties:

- Change the LayerUpdate property to **exLayerUpdateScreen**, so the entire control is shown individually on the screen, with no form behind.

In order to make your eXGauge library to display a transparent-control inside your form/dialog/window/child, you need to use the following properties:

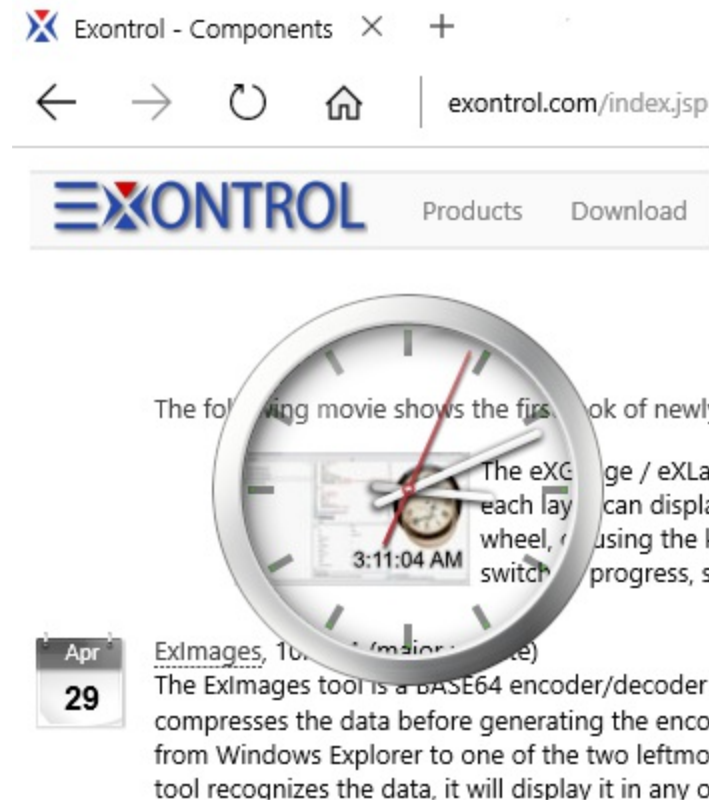
- Change the LayerUpdate property to **exLayerUpdateParent**, so the control itself ( with no nackground ) is shown on the form/dialog/parent.
- You need to add **<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}"/>**, to your manifest file as follows. The transparent-eXGauge as a child of your form, it is supported on Windows 8, and later.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1"
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
  <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
    <application>
      <supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}"/>
    </application>
  </compatibility>
</assembly>
```

The control installs the

- C:\Program Files\Exontrol\ExGauge\Sample\VB\Widget or C:\Program Files\Exontrol\ExGauge\Sample\VC\Widget sample that shows how exLayerUpdateScreenworks with LayerUpdate property.
- C:\Program Files\Exontrol\ExGauge\Sample\VC\Widget-Child sample that shows how exLayerUpdateParent works with LayerUpdate property

The following screen shot shows the control on a transparent form (**exLayerUpdateScreen**):



The following screen shot shows the transparent-control on form (**exLayerUpdateParent**):



[www.exontrol.com](http://www.exontrol.com)

The message doesn't appear in the registered version.



# property Gauge.PicturesName as String

Specifies the expression that indicates the name of the picture to be loaded on each layer.

Type	Description
String	A String value that defines the expression to specify the name of the picture to be loaded on each layer.

By default, the PicturesName property is empty. The [Picture.Name](#) / [Picture.Value](#) property is initialized by evaluating the control's PicturesName property, whose **value** keyword is replaced by the [Picture.Index](#) of the current layer.

The following properties can be used to load / import ( manually or automatically ) pictures to the layer's background:

- [PicturesPath](#) property, specifies the path to load pictures from.
- PicturesName property, specifies the expression that defines the name of the file from the PicturesPath folder to be loaded.
- [Picture.Name](#) / [Picture.Value](#) property of the Background.Picture object, defines the name of the file to be loaded ( relative, absolute, encoded or Picture object )

The [PicturesPath](#) / PicturesName properties can be used to automatically loads files from a specified folder to be displayed on the layer's background. The [Picture.Name](#) / [Picture.Value](#) property of the Picture object loads a picture / graphics to be displayed on the layer's background.

For instance,

PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob",  
defines default folder to load pictures from.

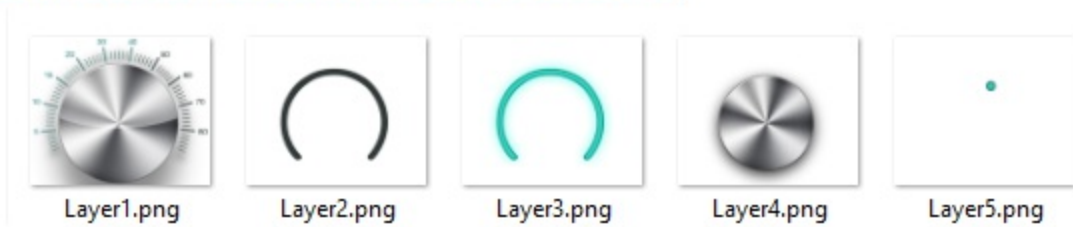
PicturesName = ""Layer` + str(value + 1) + `.png`", defines the name of the picture file to be loaded by the layer with the index / value. It defines the names as: Layer1.png for the layer with the index 0, Layer2.png for the layer with the index 1, Layer3.png for the layer with the index 2, and so on.

The PicturesName property supports the following keywords:

- **value** keyword, specifies the [Index](#) of the layer.

Also, this property supports all constants, operators and functions defined [here](#).

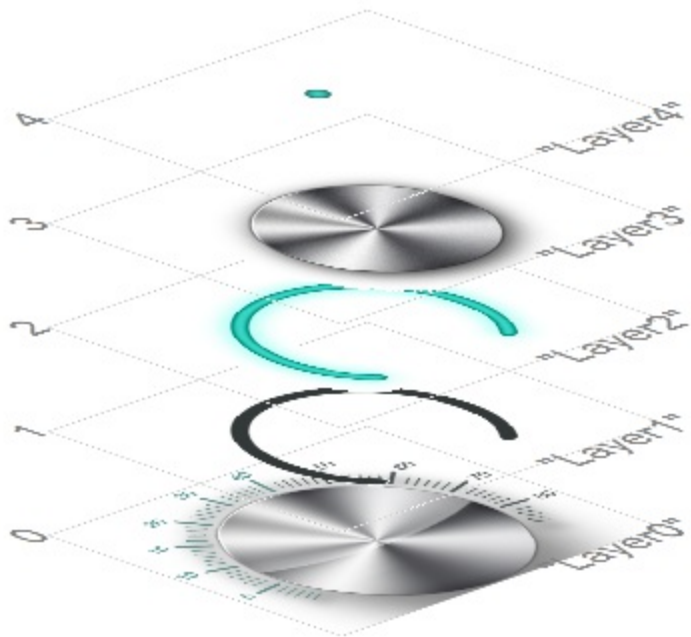
For instance, having the files Layer1.png, Layer2.png, Layer3.png, Layer4.png and Layer5.png in the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2 folder:



We can load them using the [PicturesPath](#) / PicturesName property and we get something like:



or if we decompose them, layer by layer we get:



The following samples shows how you can load automatically the Layer1.png, Layer2.png, Layer3.png, Layer4.png and Layer5.png from the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2 folder:

## VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    .EndUpdate
End With
```

## VB6

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    .EndUpdate
End With
```

## VB.NET

```
With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    .EndUpdate()
End With
```

## VB.NET for /COM

```
With AxGauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
```

```
.PicturesName = L"Layer` + int(value + 1) + `.png`"  
.Layers.Count = 5  
.EndUpdate()  
End With
```

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control  
    Library'  
  
    #import <ExGauge.dll>  
    using namespace EXGAUGELib;  
*/  
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-  
> GetControlUnknown();  
spGauge1->BeginUpdate();  
spGauge1->PutPicturesPath(L"C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2");  
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");  
spGauge1->GetLayers()->PutCount(5);  
spGauge1->EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();  
Gauge1->PicturesPath = L"C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";  
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`";  
Gauge1->Layers->Count = 5;  
Gauge1->EndUpdate();
```

## C#

```
exgauge1.BeginUpdate();
```

```
exgauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";  
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
exgauge1.Layers.Count = 5;  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.BeginUpdate();  
    Gauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 5;  
    Gauge1.EndUpdate();  
}  
</SCRIPT>  
</BODY>
```

## VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Gauge1  
        .BeginUpdate  
        .PicturesPath = "C:\\Program
```

```
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"
```

```
.PicturesName = "`Layer` + int(value + 1) + `.png`"
```

```
.Layers.Count = 5
```

```
.EndUpdate
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

## C# for /COM

```
axGauge1.BeginUpdate();
```

```
axGauge1.PicturesPath = "C:\\Program
```

```
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";
```

```
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
```

```
axGauge1.Layers.Count = 5;
```

```
axGauge1.EndUpdate();
```

## X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
```

```
;
```

```
super();
```

```
exgauge1.BeginUpdate();
```

```
exgauge1.PicturesPath("C:\\Program
```

```
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2");
```

```
exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
```

```
exgauge1.Layers().Count(5);
```

```
exgauge1.EndUpdate();
```

```
}
```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
.PicturesName = "Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.EndUpdate
endwith

```

## dBASE Plus

```

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject

```

```
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P  
  
oGauge = topparent:CONTROL_ACTIVEX1.activex  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## Visual Objects

```
oDCOCX_Exontrol1.BeginUpdate()  
oDCOCX_Exontrol1.PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oDCOCX_Exontrol1.PicturesName := "`Layer` + int(value + 1) + `.png`"  
oDCOCX_Exontrol1.Layers.Count := 5  
oDCOCX_Exontrol1.EndUpdate()
```

## PowerBuilder

```
OleObject oGauge  
  
oGauge = ole_1.Object  
oGauge.BeginUpdate()
```



```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Send ComBeginUpdate  
    Set ComPicturesPath to "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
    Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"  
    Variant voLayers  
    Get ComLayers to voLayers  
    Handle hoLayers  
    Get Create (RefClass(cComLayers)) to hoLayers  
    Set pvComObject of hoLayers to voLayers  
        Set ComCount of hoLayers to 5  
    Send Destroy to hoLayers  
    Send ComEndUpdate  
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
    LOCAL oForm  
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
    LOCAL oGauge  
  
    oForm := XbpDialog():new( AppDesktop() )  
    oForm:drawingArea:clipChildren := .T.  
    oForm:create( „{100,100}, {640,480}„, .F. )
```

```

oForm:close := {|| PostAppEvent( xbeP_Quit )}

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()
    oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 5
    oGauge:EndUpdate()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

# property Gauge.PicturesPath as String

Specifies the path to load the pictures from.

Type	Description
String	A String expression that defines the folder to load pictures from

By default, the PicturesPath property is empty. The PicturesPath property specifies the path to load the pictures from. The [PicturesName](#) property specifies the expression that indicates the name of the picture to be loaded on each layer. The [Count](#) property specifies the number of layers in the control.

The following properties can be used to load / import ( manually or automatically ) pictures to the layer's background:

- PicturesPath property, specifies the path to load pictures from.
- [PicturesName](#) property, specifies the expression that defines the name of the file from the PicturesPath folder to be loaded. By default, the [Picture.Name](#) / [Picture.Value](#) property is initialized by evaluating the control's [PicturesName](#) property, whose **value** keyword is replaced by the [Picture.Index](#) of the current layer.
- [Picture.Name](#) / [Picture.Value](#) property of the Background.Picture object, defines the name of the file to be loaded ( relative, absolute, encoded or Picture object )

The PicturesPath / [PicturesName](#) properties can be used to automatically loads files from a specified folder to be displayed on the layer's background.

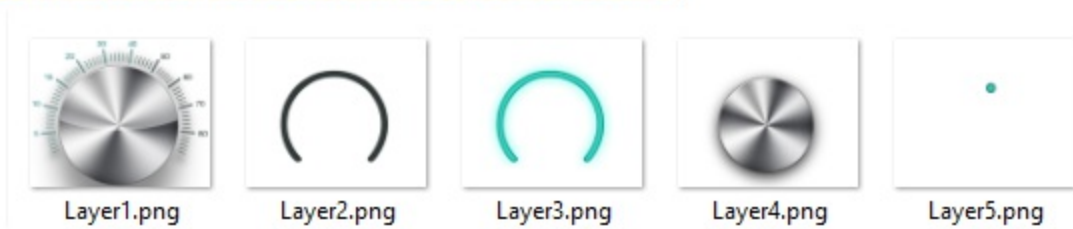
For instance,

PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob",  
defines default folder to load pictures from.

PicturesName = ""Layer` + str(value + 1) + `.png`", defines the name of the picture file to be loaded by the layer with the index / value. It defines the names as: Layer1.png for the layer with the index 0, Layer2.png for the layer with the index 1, Layer3.png for the layer with the index 2, and so on.

The [Picture.Name](#) / [Picture.Value](#) property of the Picture object loads a picture / graphics to be displayed on the layer's background.

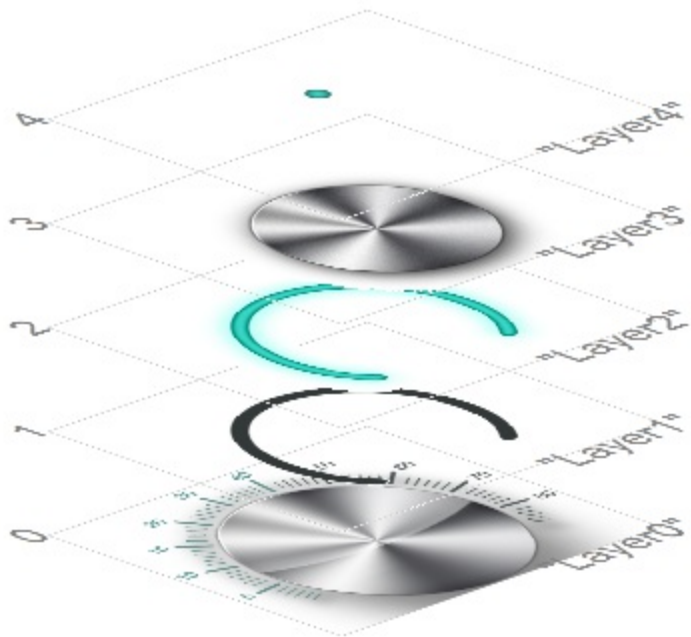
For instance, having the files Layer1.png, Layer2.png, Layer3.png, Layer4.png and Layer5.png in the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2 folder:



We can load them using the PicturesPath / [PicturesName](#) property and we get something like:



or if we decompose them, layer by layer we get:



The following samples shows how you can load automatically the Layer1.png, Layer2.png, Layer3.png, Layer4.png and Layer5.png from the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2 folder:

## VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    .EndUpdate
End With
```

## VB6

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    .EndUpdate
End With
```

## VB.NET

```
With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    .EndUpdate()
End With
```

## VB.NET for /COM

```
With AxGauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
```

```
.PicturesName = L"Layer` + int(value + 1) + `.png`"  
.Layers.Count = 5  
.EndUpdate()  
End With
```

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control  
    Library'  
  
    #import <ExGauge.dll>  
    using namespace EXGAUGELib;  
*/  
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-  
> GetControlUnknown();  
spGauge1->BeginUpdate();  
spGauge1->PutPicturesPath(L"C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2");  
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png`");  
spGauge1->GetLayers()->PutCount(5);  
spGauge1->EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();  
Gauge1->PicturesPath = L"C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";  
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`";  
Gauge1->Layers->Count = 5;  
Gauge1->EndUpdate();
```

## C#

```
exgauge1.BeginUpdate();
```

```
exgauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";  
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
exgauge1.Layers.Count = 5;  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.BeginUpdate();  
    Gauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 5;  
    Gauge1.EndUpdate();  
}  
</SCRIPT>  
</BODY>
```

## VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Gauge1  
        .BeginUpdate  
        .PicturesPath = "C:\\Program
```

```
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"
```

```
.PicturesName = "`Layer` + int(value + 1) + `.png`"
```

```
.Layers.Count = 5
```

```
.EndUpdate
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

## C# for /COM

```
axGauge1.BeginUpdate();
```

```
axGauge1.PicturesPath = "C:\\Program
```

```
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";
```

```
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
```

```
axGauge1.Layers.Count = 5;
```

```
axGauge1.EndUpdate();
```

## X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
```

```
;
```

```
super();
```

```
exgauge1.BeginUpdate();
```

```
exgauge1.PicturesPath("C:\\Program
```

```
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2");
```

```
exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
```

```
exgauge1.Layers().Count(5);
```

```
exgauge1.EndUpdate();
```

```
}
```

## Delphi 8 (.NET only)



```

with AxGauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
.PicturesName = "Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
.EndUpdate
endwith

```

## dBASE Plus

```

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject

```

```
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P  
  
oGauge = topparent:CONTROL_ACTIVEX1.activex  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## Visual Objects

```
oDCOCX_Exontrol1.BeginUpdate()  
oDCOCX_Exontrol1.PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oDCOCX_Exontrol1.PicturesName := "`Layer` + int(value + 1) + `.png`"  
oDCOCX_Exontrol1.Layers.Count := 5  
oDCOCX_Exontrol1.EndUpdate()
```

## PowerBuilder

```
OleObject oGauge  
  
oGauge = ole_1.Object  
oGauge.BeginUpdate()
```

```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Send ComBeginUpdate  
    Set ComPicturesPath to "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
    Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"  
    Variant voLayers  
    Get ComLayers to voLayers  
    Handle hoLayers  
    Get Create (RefClass(cComLayers)) to hoLayers  
    Set pvComObject of hoLayers to voLayers  
        Set ComCount of hoLayers to 5  
    Send Destroy to hoLayers  
    Send ComEndUpdate  
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
    LOCAL oForm  
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
    LOCAL oGauge  
  
    oForm := XbpDialog():new( AppDesktop() )  
    oForm:drawingArea:clipChildren := .T.  
    oForm:create( „{100,100}, {640,480}„, .F. )
```

```

oForm:close := {|| PostAppEvent( xbeP_Quit )}

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()
    oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 5
    oGauge:EndUpdate()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

# method Gauge.Refresh ()

Refreses the control.

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

# method Gauge.Replacelcon ([Icon as Variant], [Index as Variant])

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

Type	Description
Icon as Variant	<p>A Variant expression that specifies the icon to add or insert, as one of the following options:</p> <ul style="list-style-type: none"><li>• a long expression that specifies the handle of the icon (HICON)</li><li>• a string expression that indicates the path to the picture file</li><li>• a string expression that defines the picture's content encoded as BASE64 strings using the <a href="#">eXImages</a> tool</li><li>• a Picture reference, which is an object that holds image data. It is often used in controls like PictureBox, Image, or in custom controls (e.g., IPicture, IPictureDisp)</li></ul> <p>If the Icon parameter is 0, it specifies that the icon at the given Index is removed. Furthermore, setting the Index parameter to -1 removes all icons.</p> <p>By default, if the Icon parameter is not specified or is missing, a value of 0 is used.</p>
Index as Variant	<p>A long expression that defines the index of the icon to insert or remove, as follows:</p> <ul style="list-style-type: none"><li>• A zero or positive value specifies the index of the icon to insert (when Icon is non-zero) or to remove (when the Icon parameter is zero)</li><li>• A negative value clears all icons when the Icon parameter is zero</li></ul> <p>By default, if the Index parameter is not specified or is missing, a value of -1 is used.</p>
Return	Description
Long	A long expression that indicates the index of the icon in the images collection

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

collection. Also, the `Replacelcon` property can clear the images collection. Use the [Images](#) method to attach a image list to the control. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection.

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

```
i = ExGauge1.Replacelcon( 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 = ExGauge1.Replacelcon( 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:

```
ExGauge1.Replacelcon 0, i, where i specifies the index of icon removed.
```

The following VB clears the control's icons collection:

```
ExGauge1.Replacelcon 0, -1
```

# property Gauge.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 tree control. Use the [ReplaceIcon](#) method to add, remove or clear icons in the control's images collection.



# property Gauge.ShowLayers as String

Indicates the only layers to be shown on the control.

Type	Description
String	<p>A String expression that could be:</p> <ul style="list-style-type: none"><li>• "all", specifies that all visible layers are shown. The <a href="#">Visible</a> property indicates the visible layers.</li><li>• "", no layer is shown in the control, no matter of the layer's <a href="#">Visible</a> property.</li><li>• "n1,n2,n3,..." specifies the list of layers to be shown, no matter of the layer's <a href="#">Visible</a> property, where n1, n2, ... are numbers ( indicating the index of the layer to be shown ). For instance "0" specifies that just the layer with the index 0 is show, "0,1,4", indicates that just layers with the specified index are displayed.</li></ul>

By default the ShowLayers property is "all", which indicates that all visible layers in the control are shown. The ShowLayers property indicates the only layers to be shown on the control. For instance, you can use the ShowLayers property to show only a few layers within the control. The purpose can be debugging a specified layer only for instance. The [Visible](#) property shows or hides the layer.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The VisibleCount property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following properties can be used to add layers within the control:

- [Count](#) property, adds layers to the control
- [Add](#) method, adds a new layer to the control.

The following properties can be used to remove layers within the control:

- [Count](#) property, removes layers from the control. For instance, Count property on 0, removes all layers from the control.
- [Clear](#) removes all layers from the control.

- [Remove](#) method, removes a layer from the control based on its index or key.

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

Shows the specified tooltip at given position.

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

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

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

X as Variant

---

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

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

Y as Variant

---

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the [MouseMove](#) event. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

For instance:

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

to its default position

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

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

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

- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number;** ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>subscript**" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>superscript**" displays the text such as: Text with superscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The **<font>** HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The **<gra>** with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "**<font ;18><gra FFFFFFFF;1;1>gradient-center</gra></font>**" generates the following picture:

gradient-center

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

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

<fgcolor=FFFFFF>outlined</fgcolor></out></font>" generates the following picture:

outlined

- <sha rrggbb;width;offset> ... </sha> define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

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

outline anti-aliasing

# property Gauge.Template as String

Specifies the control's template.

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

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

The Exontrol's [eXHelper](#) tool helps you to find easy and quickly the answers and the source code for your questions regarding the usage of our UI components.

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.*

For instance, the following script:

```
PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
PicturesName = "`Layer` + str(value + 1) + `.png`"
```

```
Layers.Count = 10
```

generates:



# property Gauge.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 gauge.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

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

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

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

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

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

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

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

The x-script may use constant expressions as follows:

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

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

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



# property Gauge.TimerInterval as Long

Returns or sets the number of milliseconds between calls of control's Timer event.

Type	Description
Long	A Long expression that specifies the number of milliseconds between calls of control's Timer event.

By default, the TimerInterval property is 0, which indicates that no [Timer](#) event occurs. The TimerInterval property returns or sets the number of milliseconds between calls of control's [Timer](#) event. You can use the [Timer](#) event to perform different actions on any layer when a specified time elapsed. For instance, you can rotate the layer every second, or any dial of a clock, and so on. The [FormatABC](#) method formats the A,B,C values based on the giving expression and returns the result. For instance, the FormatABC("date(`now`)") gets the current time.

You can use any of the following properties to update the layer:

- [Value](#), specifies the layer's value.
- [OffsetX](#), specifies a value that indicates x-offset of the layer.
- [OffsetY](#), indicates a value that indicates y-offset of the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [Clip](#), to clip any layer

The [Change](#) event occurs when the layer's value is changed.

The following sample shows how you can display a clock:



## VBA (MS Access, Excell...)

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)
    With Gauge1
```

```

.Layers.Item("sec").Value = Gauge1.Value
.Layers.Item("min").Value = Gauge1.Value
.Layers.Item("hour").Value = Gauge1.Value
End With
End Sub

' Timer event - Occurs when the interval elapses.
Private Sub Gauge1_Timer(ByVal TickCount As Long)
    With Gauge1
        .Value = .FormatABC("value + 1/24/60/60",.Value)
    End With
End Sub

With Gauge1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
    .DefaultLayer(185) = 2
    .Layers.Count = 4
    With .Layers.Item(0)
        .Background.Picture.Name = "vista_clock.png"
    End With
    With .Layers.Item(1)
        .Position = 3
        .Key = "sec"
        .OnDrag = 2
        .Selectable = False
        .Background.Picture.Name = "second-hand.png"
        .ValueToRotateAngle = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " & _
"floor(=:1)) * 60 )) - floor(=:2) ) * 360"
        .RotateAngleToValue = "value / 360 / 24 / 60"
    End With
    With .Layers.Item(2)
        .Position = 2
        .Key = "min"
        .OnDrag = 2
        .Selectable = False

```

```

        .Background.Picture.Name = "Minute.png"
        .ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" & _
"=:1)) * 360"
        .RotateAngleToValue = "value / 360 / 24 / 60"
End With
With .Layers.Item(3)
    .Position = 1
    .Key = "hour"
    .OnDrag = 2
    .Selectable = False
    .Background.Picture.Name = "Hour.png"
    .ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
    .RotateAngleToValue = "value / 360 * 0.5"
End With
.LayerOfValue = 3
.Value = .FormatABC("date(now)")
.TimerInterval = 1000
End With

```

## VB6

*' Change event - Occurs when the layer's value is changed.*

```

Private Sub Gauge1_Change(ByVal Layer As Long)
    With Gauge1
        .Layers.Item("sec").Value = Gauge1.Value
        .Layers.Item("min").Value = Gauge1.Value
        .Layers.Item("hour").Value = Gauge1.Value
    End With
End Sub

```

*' Timer event - Occurs when the interval elapses.*

```

Private Sub Gauge1_Timer(ByVal TickCount As Long)
    With Gauge1
        .Value = .FormatABC("value + 1/24/60/60",.Value)
    End With

```

End Sub

With Gauge1

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"

.DefaultLayer(exDefLayerRotateType) = 2

.Layers.Count = 4

With .Layers.Item(0)

.Background.Picture.Name = "vista\_clock.png"

End With

With .Layers.Item(1)

.Position = 3

.Key = "sec"

.OnDrag = exDoRotate

.Selectable = False

.Background.Picture.Name = "second-hand.png"

.ValueToRotateAngle = "((2:=(((1:= ( ( 0:=(value < 0 ? floor(value) + 1 - value :  
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) \* 24 )) - " & \_  
"floor(=:1)) \* 60 )) - floor(=:2) ) \* 360"

.RotateAngleToValue = "value / 360 / 24 / 60"

End With

With .Layers.Item(2)

.Position = 2

.Key = "min"

.OnDrag = exDoRotate

.Selectable = False

.Background.Picture.Name = "Minute.png"

.ValueToRotateAngle = "((1:= ( ( 0:=(value < 0 ? floor(value) + 1 - value : value -  
floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) \* 24 )) - floor(" & \_  
"=:1)) \* 360"

.RotateAngleToValue = "value / 360 / 24 / 60"

End With

With .Layers.Item(3)

.Position = 1

.Key = "hour"

.OnDrag = exDoRotate

.Selectable = False

```

        .Background.Picture.Name = "Hour.png"
        .ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value
- floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
        .RotateAngleToValue = "value / 360 * 0.5"
    End With
    .LayerOfValue = 3
    .Value = .FormatABC("date(now)")
    .TimerInterval = 1000
End With

```

## VB.NET

*' Change event - Occurs when the layer's value is changed.*

```

Private Sub Exgauge1_Change(ByVal sender As System.Object,ByVal Layer As Integer)
Handles Exgauge1.Change
    With Exgauge1
        .Layers.Item("sec").Value = Exgauge1.Value
        .Layers.Item("min").Value = Exgauge1.Value
        .Layers.Item("hour").Value = Exgauge1.Value
    End With
End Sub

```

*' Timer event - Occurs when the interval elapses.*

```

Private Sub Exgauge1_Timer(ByVal sender As System.Object,ByVal TickCount As
Integer) Handles Exgauge1.Timer
    With Exgauge1
        .Value = .FormatABC("value + 1/24/60/60",.Value)
    End With
End Sub

```

With Exgauge1

```

    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"

```

```

.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateTy

```

```

    .Layers.Count = 4

```

```

With .Layers.Item(0)
    .Background.Picture.Name = "vista_clock.png"
End With
With .Layers.Item(1)
    .Position = 3
    .Key = "sec"
    .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
    .Selectable = False
    .Background.Picture.Name = "second-hand.png"
    .ValueToRotateAngle = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " & _
"floor(=:1)) * 60 )) - floor(=:2) ) * 360"
    .RotateAngleToValue = "value / 360 / 24 / 60"
End With
With .Layers.Item(2)
    .Position = 2
    .Key = "min"
    .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
    .Selectable = False
    .Background.Picture.Name = "Minute.png"
    .ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value -
floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" & _
"=:1)) * 360"
    .RotateAngleToValue = "value / 360 / 24 / 60"
End With
With .Layers.Item(3)
    .Position = 1
    .Key = "hour"
    .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
    .Selectable = False
    .Background.Picture.Name = "Hour.png"
    .ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value
- floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
    .RotateAngleToValue = "value / 360 * 0.5"
End With
.LayerOfValue = 3
.Value = .FormatABC("date(`now`)")

```

**.TimerInterval** = 1000

End With

## VB.NET for /COM

*' Change event - Occurs when the layer's value is changed.*

Private Sub AxGauge1\_Change(ByVal sender As System.Object, ByVal e As AxEXGAUGELib.\_IGaugeEvents\_ChangeEvent) Handles AxGauge1.Change

With AxGauge1

.Layers.Item("sec").Value = AxGauge1.Value

.Layers.Item("min").Value = AxGauge1.Value

.Layers.Item("hour").Value = AxGauge1.Value

End With

End Sub

*' Timer event - Occurs when the interval elapses.*

Private Sub AxGauge1\_Timer(ByVal sender As System.Object, ByVal e As AxEXGAUGELib.\_IGaugeEvents\_TimerEvent) Handles AxGauge1.Timer

With AxGauge1

.Value = .FormatABC("value + 1/24/60/60",.Value)

End With

End Sub

With AxGauge1

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"

.set\_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,2)

.Layers.Count = 4

With .Layers.Item(0)

.Background.Picture.Name = "vista\_clock.png"

End With

With .Layers.Item(1)

.Position = 3

.Key = "sec"

.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate

.Selectable = False

.Background.Picture.Name = "second-hand.png"

```

        .ValueToRotateAngle = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " & _
        "floor(=:1)) * 60 )) - floor(=:2) ) * 360"
        .RotateAngleToValue = "value / 360 / 24 / 60"
    End With
    With .Layers.Item(2)
        .Position = 2
        .Key = "min"
        .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
        .Selectable = False
        .Background.Picture.Name = "Minute.png"
        .ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" & _
        "=:1)) * 360"
        .RotateAngleToValue = "value / 360 / 24 / 60"
    End With
    With .Layers.Item(3)
        .Position = 1
        .Key = "hour"
        .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
        .Selectable = False
        .Background.Picture.Name = "Hour.png"
        .ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
        .RotateAngleToValue = "value / 360 * 0.5"
    End With
    .LayerOfValue = 3
    .Value = .FormatABC("date(`now`)")
    .TimerInterval = 1000
End With

```

**C++**

```

// Change event - Occurs when the layer's value is changed.
void OnChangeGauge1(long Layer)
{
    /*

```



*Copy and paste the following directives to your header file as it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control Library'*

```
#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->GetLayers()->GetItem("sec")->PutValue(spGauge1->GetValue());
spGauge1->GetLayers()->GetItem("min")->PutValue(spGauge1->GetValue());
spGauge1->GetLayers()->GetItem("hour")->PutValue(spGauge1->GetValue());
}

// Timer event - Occurs when the interval elapses.
void OnTimerGauge1(long TickCount)
{
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    spGauge1->PutValue(spGauge1->FormatABC(L"value + 1/24/60/60",spGauge1-
> GetValue(),vtMissing,vtMissing));
}
```

```
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Clock");
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerRotateType,long(2));
spGauge1->GetLayers()->PutCount(4);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(long(0));
var_Layer->GetBackground()->GetPicture()->PutName("vista_clock.png");
EXGAUGELib::ILayerPtr var_Layer1 = spGauge1->GetLayers()->GetItem(long(1));
var_Layer1->PutPosition(3);
var_Layer1->PutKey("sec");
var_Layer1->PutOnDrag(EXGAUGELib::exDoRotate);
var_Layer1->PutSelectable(VARIANT_FALSE);
var_Layer1->GetBackground()->GetPicture()->PutName("second-hand.png");
var_Layer1->PutValueToRotateAngle(_bstr_t("((2:=(((1:=((0:=(value < 0 ?
```

```

floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - "
+
"floor(=:1)) * 60 )) - floor(=:2) ) * 360");
var_Layer1->PutRotateAngleToValue(L"value / 360 / 24 / 60");
EXGAUGELib::ILayerPtr var_Layer2 = spGauge1->GetLayers()->GetItem(long(2));
var_Layer2->PutPosition(2);
var_Layer2->PutKey("min");
var_Layer2->PutOnDrag(EXGAUGELib::exDoRotate);
var_Layer2->PutSelectable(VARIANT_FALSE);
var_Layer2->GetBackground()->GetPicture()->PutName("Minute.png");
var_Layer2->PutValueToRotateAngle(_bstr_t("((1:=( ( 0:=(value < 0 ? floor(value) +
1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" +
"=:1)) * 360");
var_Layer2->PutRotateAngleToValue(L"value / 360 / 24 / 60");
EXGAUGELib::ILayerPtr var_Layer3 = spGauge1->GetLayers()->GetItem(long(3));
var_Layer3->PutPosition(1);
var_Layer3->PutKey("hour");
var_Layer3->PutOnDrag(EXGAUGELib::exDoRotate);
var_Layer3->PutSelectable(VARIANT_FALSE);
var_Layer3->GetBackground()->GetPicture()->PutName("Hour.png");
var_Layer3->PutValueToRotateAngle(L"2 * 360 * ( 0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )");
var_Layer3->PutRotateAngleToValue(L"value / 360 * 0.5");
spGauge1->PutLayerOfValue(3);
spGauge1->PutValue(spGauge1-
>FormatABC(L"date(now)",vtMissing,vtMissing,vtMissing));
spGauge1->PutTimerInterval(1000);

```

## C++ Builder

```

// Change event - Occurs when the layer's value is changed.
void __fastcall TForm1::Gauge1Change(TObject *Sender,long Layer)
{
    Gauge1->Layers->get_Item(TVariant("sec"))->set_Value(TVariant(Gauge1-
>get_Value()));
    Gauge1->Layers->get_Item(TVariant("min"))->set_Value(TVariant(Gauge1-

```

```
>get_Value());
    Gauge1->Layers->get_Item(TVariant("hour"))->set_Value(TVariant(Gauge1-
>get_Value()));
}
```

*// Timer event - Occurs when the interval elapses.*

```
void __fastcall TForm1::Gauge1Timer(TObject *Sender,long TickCount)
{
    Gauge1->set_Value(TVariant(Gauge1->FormatABC(L"value +
1/24/60/60",TVariant(Gauge1->get_Value()),TNoParam(),TNoParam())));
}
```

```
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Clock";
```

```
Gauge1-
```

```
>DefaultLayer[Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerRotateType] =
TVariant(2);
```

```
Gauge1->Layers->Count = 4;
```

```
Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(0));
```

```
    var_Layer->Background->Picture->set_Name(TVariant("vista_clock.png"));
```

```
Exgaugelib_tlb::ILayerPtr var_Layer1 = Gauge1->Layers->get_Item(TVariant(1));
```

```
    var_Layer1->Position = 3;
```

```
    var_Layer1->set_Key(TVariant("sec"));
```

```
    var_Layer1->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
```

```
    var_Layer1->Selectable = false;
```

```
    var_Layer1->Background->Picture->set_Name(TVariant("second-hand.png"));
```

```
    var_Layer1->ValueToRotateAngle = TVariant(String("((2:=(((1:= ( ( 0:=(value < 0 ?
floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - ")
```

```
+
```

```
"floor(=:1)) * 60 )) - floor(=:2) ) * 360");
```

```
    var_Layer1->RotateAngleToValue = L"value / 360 / 24 / 60";
```

```
Exgaugelib_tlb::ILayerPtr var_Layer2 = Gauge1->Layers->get_Item(TVariant(2));
```

```
    var_Layer2->Position = 2;
```

```
    var_Layer2->set_Key(TVariant("min"));
```

```
    var_Layer2->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
```

```
    var_Layer2->Selectable = false;
```

```
    var_Layer2->Background->Picture->set_Name(TVariant("Minute.png"));
```

```

var_Layer2->ValueToRotateAngle = TVariant(String("((1:=( ( (0:=(value < 0 ?
floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) -
floor(") +
"=:1)) * 360");
var_Layer2->RotateAngleToValue = L"value / 360 / 24 / 60";
Exgaugelib_tlb::ILayerPtr var_Layer3 = Gauge1->Layers->get_Item(TVariant(3));
var_Layer3->Position = 1;
var_Layer3->set_Key(TVariant("hour"));
var_Layer3->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
var_Layer3->Selectable = false;
var_Layer3->Background->Picture->set_Name(TVariant("Hour.png"));
var_Layer3->ValueToRotateAngle = L"2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )";
var_Layer3->RotateAngleToValue = L"value / 360 * 0.5";
Gauge1->LayerOfValue = 3;
Gauge1->set_Value(TVariant(Gauge1-
>FormatABC(L"date(now)",TNoParam(),TNoParam(),TNoParam())));
Gauge1->TimerInterval = 1000;

```

## C#

```

// Change event - Occurs when the layer's value is changed.
private void exgauge1_Change(object sender,int Layer)
{
    exgauge1.Layers["sec"].Value = exgauge1.Value;
    exgauge1.Layers["min"].Value = exgauge1.Value;
    exgauge1.Layers["hour"].Value = exgauge1.Value;
}
//this.exgauge1.Change += new
exontrol.EXGAUGELib.exg2antt.ChangeEventHandler(this.exgauge1_Change);

// Timer event - Occurs when the interval elapses.
private void exgauge1_Timer(object sender,int TickCount)
{
    exgauge1.Value = exgauge1.FormatABC("value +
1/24/60/60",exgauge1.Value,null,null);
}

```

```

}
//this.exgauge1.Timer += new
exontrol.EXGAUGELib.exg2antt.TimerEventHandler(this.exgauge1_Timer);

exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Clock";
exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLay

exgauge1.Layers.Count = 4;
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[0];
    var_Layer.Background.Picture.Name = "vista_clock.png";
exontrol.EXGAUGELib.Layer var_Layer1 = exgauge1.Layers[1];
    var_Layer1.Position = 3;
    var_Layer1.Key = "sec";
    var_Layer1.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.Selectable = false;
    var_Layer1.Background.Picture.Name = "second-hand.png";
    var_Layer1.ValueToRotateAngle = "((2:=(((1:=((0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " +
"floor(=:1)) * 60 )) - floor(=:2) ) * 360";
    var_Layer1.RotateAngleToValue = "value / 360 / 24 / 60";
exontrol.EXGAUGELib.Layer var_Layer2 = exgauge1.Layers[2];
    var_Layer2.Position = 2;
    var_Layer2.Key = "min";
    var_Layer2.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer2.Selectable = false;
    var_Layer2.Background.Picture.Name = "Minute.png";
    var_Layer2.ValueToRotateAngle = "((1:=((0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" +
"=:1)) * 360";
    var_Layer2.RotateAngleToValue = "value / 360 / 24 / 60";
exontrol.EXGAUGELib.Layer var_Layer3 = exgauge1.Layers[3];
    var_Layer3.Position = 1;
    var_Layer3.Key = "hour";
    var_Layer3.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer3.Selectable = false;
    var_Layer3.Background.Picture.Name = "Hour.png";

```

```

var_Layer3.ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )";
var_Layer3.RotateAngleToValue = "value / 360 * 0.5";
exgauge1.LayerOfValue = 3;
exgauge1.Value = exgauge1.FormatABC("date('now')",null,null,null);
exgauge1.TimerInterval = 1000;

```

## JScript/JavaScript

```

<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="Change(Layer)" LANGUAGE="JScript">
    Gauge1.Layers.Item("sec").Value = Gauge1.Value;
    Gauge1.Layers.Item("min").Value = Gauge1.Value;
    Gauge1.Layers.Item("hour").Value = Gauge1.Value;
</SCRIPT>

<SCRIPT FOR="Gauge1" EVENT="Timer(TickCount)" LANGUAGE="JScript">
    Gauge1.Value = Gauge1.FormatABC("value + 1/24/60/60",Gauge1.Value,null,null);
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Clock";
    Gauge1.DefaultLayer(185) = 2;
    Gauge1.Layers.Count = 4;
    var var_Layer = Gauge1.Layers.Item(0);
    var_Layer.Background.Picture.Name = "vista_clock.png";
    var var_Layer1 = Gauge1.Layers.Item(1);
    var_Layer1.Position = 3;
    var_Layer1.Key = "sec";
    var_Layer1.OnDrag = 2;

```

```

var_Layer1.Selectable = false;
var_Layer1.Background.Picture.Name = "second-hand.png";
var_Layer1.ValueToRotateAngle = "((2:=(((1:= ( ( 0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " +
"floor(=:1)) * 60 )) - floor(=:2) ) * 360";
var_Layer1.RotateAngleToValue = "value / 360 / 24 / 60";
var var_Layer2 = Gauge1.Layers.Item(2);
var_Layer2.Position = 2;
var_Layer2.Key = "min";
var_Layer2.OnDrag = 2;
var_Layer2.Selectable = false;
var_Layer2.Background.Picture.Name = "Minute.png";
var_Layer2.ValueToRotateAngle = "((1:= ( ( 0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" +
"=:1)) * 360";
var_Layer2.RotateAngleToValue = "value / 360 / 24 / 60";
var var_Layer3 = Gauge1.Layers.Item(3);
var_Layer3.Position = 1;
var_Layer3.Key = "hour";
var_Layer3.OnDrag = 2;
var_Layer3.Selectable = false;
var_Layer3.Background.Picture.Name = "Hour.png";
var_Layer3.ValueToRotateAngle = "2 * 360 * ( 0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )";
var_Layer3.RotateAngleToValue = "value / 360 * 0.5";
Gauge1.LayerOfValue = 3;
Gauge1.Value = Gauge1.FormatABC("date(now)",null,null,null);
Gauge1.TimerInterval = 1000;
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">

```

```
Function Gauge1_Change(Layer)
```

```
    With Gauge1
```

```
        .Layers.Item("sec").Value = Gauge1.Value
```

```
        .Layers.Item("min").Value = Gauge1.Value
```

```
        .Layers.Item("hour").Value = Gauge1.Value
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

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

```
Function Gauge1_Timer(TickCount)
```

```
    With Gauge1
```

```
        .Value = .FormatABC("value + 1/24/60/60",.Value)
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
```

```
id="Gauge1"></OBJECT>
```

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

```
Function Init()
```

```
    With Gauge1
```

```
        .PicturesPath = "C:\Program
```

```
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
```

```
        .DefaultLayer(185) = 2
```

```
        .Layers.Count = 4
```

```
        With .Layers.Item(0)
```

```
            .Background.Picture.Name = "vista_clock.png"
```

```
        End With
```

```
        With .Layers.Item(1)
```

```
            .Position = 3
```

```
            .Key = "sec"
```

```
            .OnDrag = 2
```

```
            .Selectable = False
```

```
            .Background.Picture.Name = "second-hand.png"
```

```
            .ValueToRotateAngle = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
```



```

value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " & _
"floor(=:1)) * 60 )) - floor(=:2)) * 360"
    .RotateAngleToValue = "value / 360 / 24 / 60"
End With
With .Layers.Item(2)
    .Position = 2
    .Key = "min"
    .OnDrag = 2
    .Selectable = False
    .Background.Picture.Name = "Minute.png"
    .ValueToRotateAngle = "((1:= ( ( 0:=(value < 0 ? floor(value) + 1 - value : value
- floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" & _
"=:1)) * 360"
    .RotateAngleToValue = "value / 360 / 24 / 60"
End With
With .Layers.Item(3)
    .Position = 1
    .Key = "hour"
    .OnDrag = 2
    .Selectable = False
    .Background.Picture.Name = "Hour.png"
    .ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
    .RotateAngleToValue = "value / 360 * 0.5"
End With
.LayerOfValue = 3
.Value = .FormatABC("date(`now`)")
.TimerInterval = 1000
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```
// Change event - Occurs when the layer's value is changed.
```

```

private void axGauge1_Change(object sender,
AxEXGAUGELib._IGaugeEvents_ChangeEvent e)
{
    axGauge1.Layers["sec"].Value = axGauge1.Value;
    axGauge1.Layers["min"].Value = axGauge1.Value;
    axGauge1.Layers["hour"].Value = axGauge1.Value;
}
//this.axGauge1.Change += new
AxEXGAUGELib._IGaugeEvents_ChangeEventHandler(this.axGauge1_Change);

// Timer event - Occurs when the interval elapses.
private void axGauge1_Timer(object sender,
AxEXGAUGELib._IGaugeEvents_TimerEvent e)
{
    axGauge1.Value = axGauge1.FormatABC("value +
1/24/60/60",axGauge1.Value,null,null);
}
//this.axGauge1.Timer += new
AxEXGAUGELib._IGaugeEvents_TimerEventHandler(this.axGauge1_Timer);

axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Clock";
axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotate

axGauge1.Layers.Count = 4;
EXGAUGELib.Layer var_Layer = axGauge1.Layers[0];
    var_Layer.Background.Picture.Name = "vista_clock.png";
EXGAUGELib.Layer var_Layer1 = axGauge1.Layers[1];
    var_Layer1.Position = 3;
    var_Layer1.Key = "sec";
    var_Layer1.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.Selectable = false;
    var_Layer1.Background.Picture.Name = "second-hand.png";
    var_Layer1.ValueToRotateAngle = "((2:=(((1:= ( ( 0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - " +
"floor(=:1)) * 60 )) - floor(=:2) ) * 360";
    var_Layer1.RotateAngleToValue = "value / 360 / 24 / 60";

```

```

EXGAUGELib.Layer var_Layer2 = axGauge1.Layers[2];
var_Layer2.Position = 2;
var_Layer2.Key = "min";
var_Layer2.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
var_Layer2.Selectable = false;
var_Layer2.Background.Picture.Name = "Minute.png";
var_Layer2.ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(" +
"=:1)) * 360";
var_Layer2.RotateAngleToValue = "value / 360 / 24 / 60";
EXGAUGELib.Layer var_Layer3 = axGauge1.Layers[3];
var_Layer3.Position = 1;
var_Layer3.Key = "hour";
var_Layer3.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
var_Layer3.Selectable = false;
var_Layer3.Background.Picture.Name = "Hour.png";
var_Layer3.ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )";
var_Layer3.RotateAngleToValue = "value / 360 * 0.5";
axGauge1.LayerOfValue = 3;
axGauge1.Value = axGauge1.FormatABC("date(`now`)",null,null,null);
axGauge1.TimerInterval = 1000;

```

## X++ (Dynamics Ax 2009)

```

// Change event - Occurs when the layer's value is changed.
void onEvent_Change(int _Layer)
{
    COM com_Layer;
    anytype var_Layer;
    ;
    var_Layer = COM::createFromObject(exgauge1.Layers()).Item("sec"); com_Layer =
var_Layer;
    com_Layer.Value(exgauge1.Value());
    var_Layer = COM::createFromObject(exgauge1.Layers()).Item("min"); com_Layer =
var_Layer;

```

```

    com_Layer.Value(exgauge1.Value());
    var_Layer = COM::createFromObject(exgauge1.Layers()).Item("hour"); com_Layer =
var_Layer;
    com_Layer.Value(exgauge1.Value());
}

// Timer event - Occurs when the interval elapses.
void onEvent_Timer(int _TickCount)
{
    ;
    exgauge1.Value(exgauge1.FormatABC("value + 1/24/60/60",exgauge1.Value()));
}

public void init()
{
    COM
com_Background,com_Layer,com_Layer1,com_Layer2,com_Layer3,com_Picture;
    anytype var_Background,var_Layer,var_Layer1,var_Layer2,var_Layer3,var_Picture;
    str var_s,var_s1;
    ;

    super();

    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Clock");

exgauge1.DefaultLayer(185/*exDefLayerRotateType*/,COMVariant::createFromInt(2));
    exgauge1.Layers().Count(4);
    var_Layer =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(0));
com_Layer = var_Layer;
    var_Background = COM::createFromObject(com_Layer.Background());
com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
    com_Picture.Name("vista_clock.png");
    var_Layer1 =

```

```

COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(1));
com_Layer1 = var_Layer1;
    com_Layer1.Position(3);
    com_Layer1.Key("sec");
    com_Layer1.OnDrag(2/*exDoRotate*/);
    com_Layer1.Selectable(false);
    var_Background = COM::createFromObject(com_Layer1.Background());
com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
    com_Picture.Name("second-hand.png");
    var_s = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value -
floor(value))) < 0.5 ? :=:0 : (0:= (=:0 - 0.5)) ) * 24 )) - f";
    var_s = var_s + "loor(=:1)) * 60 )) - floor(=:2) ) * 360";
    com_Layer1.ValueToRotateAngle(var_s);
    com_Layer1.RotateAngleToValue("value / 360 / 24 / 60");
var_Layer2 =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(2));
com_Layer2 = var_Layer2;
    com_Layer2.Position(2);
    com_Layer2.Key("min");
    com_Layer2.OnDrag(2/*exDoRotate*/);
    com_Layer2.Selectable(false);
    var_Background = COM::createFromObject(com_Layer2.Background());
com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
    com_Picture.Name("Minute.png");
    var_s1 = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) <
0.5 ? :=:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=";
    var_s1 = var_s1 + ":1)) * 360";
    com_Layer2.ValueToRotateAngle(var_s1);
    com_Layer2.RotateAngleToValue("value / 360 / 24 / 60");
var_Layer3 =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(3));
com_Layer3 = var_Layer3;
    com_Layer3.Position(1);

```

```

com_Layer3.Key("hour");
com_Layer3.OnDrag(2/*exDoRotate*/);
com_Layer3.Selectable(false);
var_Background = COM::createFromObject(com_Layer3.Background());
com_Background = var_Background;
var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
com_Picture.Name("Hour.png");
com_Layer3.ValueToRotateAngle("2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )");
com_Layer3.RotateAngleToValue("value / 360 * 0.5");
exgauge1.LayerOfValue(3);
exgauge1.Value(exgauge1.FormatABC("date(`now`)"));
exgauge1.TimerInterval(1000);
}

```

## Delphi 8 (.NET only)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.AxGauge1_Change(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_ChangeEvent);
begin
  with AxGauge1 do
  begin
    Layers.Item['sec'].Value := AxGauge1.Value;
    Layers.Item['min'].Value := AxGauge1.Value;
    Layers.Item['hour'].Value := AxGauge1.Value;
  end
end;

// Timer event - Occurs when the interval elapses.
procedure TForm1.AxGauge1_Timer(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_TimerEvent);
begin
  with AxGauge1 do
  begin
    Value := FormatABC('value + 1/24/60/60',Value,Nil,Nil);
  end
end;

```

```

end
end;

with AxGauge1 do
begin
  PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock';

set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,TObje

  Layers.Count := 4;
  with Layers.Item[TObject(0)] do
  begin
    Background.Picture.Name := 'vista_clock.png';
  end;
  with Layers.Item[TObject(1)] do
  begin
    Position := 3;
    Key := 'sec';
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
    Selectable := False;
    Background.Picture.Name := 'second-hand.png';
    ValueToRotateAngle := '((2:=(((1:=( ( 0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - f' +
'loor(=:1)) * 60 )) - floor(=:2) ) * 360';
    RotateAngleToValue := 'value / 360 / 24 / 60';
  end;
  with Layers.Item[TObject(2)] do
  begin
    Position := 2;
    Key := 'min';
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
    Selectable := False;
    Background.Picture.Name := 'Minute.png';
    ValueToRotateAngle := '((1:=( ( 0:=(value < 0 ? floor(value) + 1 - value : value -
floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=' +
':1)) * 360';

```

```

    RotateAngleToValue := 'value / 360 / 24 / 60';
end;
with Layers.Item[TObject(3)] do
begin
    Position := 1;
    Key := 'hour';
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
    Selectable := False;
    Background.Picture.Name := 'Hour.png';
    ValueToRotateAngle := '2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value
- floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )';
    RotateAngleToValue := 'value / 360 * 0.5';
end;
LayerOfValue := 3;
Value := FormatABC('date(now)',Nil,Nil,Nil);
TimerInterval := 1000;
end

```

## Delphi (standard)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.Gauge1Change(ASender: TObject; Layer : Integer);
begin
    with Gauge1 do
    begin
        Layers.Item['sec'].Value := Gauge1.Value;
        Layers.Item['min'].Value := Gauge1.Value;
        Layers.Item['hour'].Value := Gauge1.Value;
    end
end;

// Timer event - Occurs when the interval elapses.
procedure TForm1.Gauge1Timer(ASender: TObject; TickCount : Integer);
begin
    with Gauge1 do
    begin
        Value := FormatABC('value + 1/24/60/60',Value,Null,Null);
    end
end;

```



```

end
end;

with Gauge1 do
begin
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock';
    DefaultLayer[EXGAUGELib_TLB.exDefLayerRotateType] := OleVariant(2);
    Layers.Count := 4;
    with Layers.Item[OleVariant(0)] do
    begin
        Background.Picture.Name := 'vista_clock.png';
    end;
    with Layers.Item[OleVariant(1)] do
    begin
        Position := 3;
        Key := 'sec';
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        Selectable := False;
        Background.Picture.Name := 'second-hand.png';
        ValueToRotateAngle := '((2:=(((1:=((0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value)))) < 0.5 ? =:0 : (0:= (=:0 - 0.5))) * 24 )) - f' +
'loor(=:1)) * 60 )) - floor(=:2) ) * 360';
        RotateAngleToValue := 'value / 360 / 24 / 60';
    end;
    with Layers.Item[OleVariant(2)] do
    begin
        Position := 2;
        Key := 'min';
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        Selectable := False;
        Background.Picture.Name := 'Minute.png';
        ValueToRotateAngle := '((1:=((0:=(value < 0 ? floor(value) + 1 - value : value -
floor(value)))) < 0.5 ? =:0 : (0:= (=:0 - 0.5))) * 24 )) - floor(=' +
':1)) * 360';
        RotateAngleToValue := 'value / 360 / 24 / 60';
    end;
end;

```

```

with Layers.Item[OleVariant(3)] do
begin
    Position := 1;
    Key := 'hour';
    OnDrag := EXGAUGELib_TLB.exDoRotate;
    Selectable := False;
    Background.Picture.Name := 'Hour.png';
    ValueToRotateAngle := '2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value
- floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )';
    RotateAngleToValue := 'value / 360 * 0.5';
end;
LayerOfValue := 3;
Value := FormatABC('date(now)',Null,Null,Null);
TimerInterval := 1000;
end

```

## VFP

*\*\*\* Change event - Occurs when the layer's value is changed. \*\*\**

LPARAMETERS Layer

```

with thisform.Gauge1
    .Layers.Item("sec").Value = thisform.Gauge1.Value
    .Layers.Item("min").Value = thisform.Gauge1.Value
    .Layers.Item("hour").Value = thisform.Gauge1.Value
endwith

```

*\*\*\* Timer event - Occurs when the interval elapses. \*\*\**

LPARAMETERS TickCount

```

with thisform.Gauge1
    .Value = .FormatABC("value + 1/24/60/60",.Value)
endwith

```

with thisform.Gauge1

```

    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
    .Object.DefaultLayer(185) = 2
    .Layers.Count = 4

```

```

with .Layers.Item(0)
    .Background.Picture.Name = "vista_clock.png"
endwith
with .Layers.Item(1)
    .Position = 3
    .Key = "sec"
    .OnDrag = 2
    .Selectable = .F.
    .Background.Picture.Name = "second-hand.png"
    var_s = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value -
floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - f"
    var_s = var_s + "loor(=:1)) * 60 )) - floor(=:2) ) * 360"
    .ValueToRotateAngle = var_s
    .RotateAngleToValue = "value / 360 / 24 / 60"
endwith
with .Layers.Item(2)
    .Position = 2
    .Key = "min"
    .OnDrag = 2
    .Selectable = .F.
    .Background.Picture.Name = "Minute.png"
    var_s1 = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) <
0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(="
    var_s1 = var_s1 + ":1)) * 360"
    .ValueToRotateAngle = var_s1
    .RotateAngleToValue = "value / 360 / 24 / 60"
endwith
with .Layers.Item(3)
    .Position = 1
    .Key = "hour"
    .OnDrag = 2
    .Selectable = .F.
    .Background.Picture.Name = "Hour.png"
    .ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 - value : value
- floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
    .RotateAngleToValue = "value / 360 * 0.5"
endwith

```

```
.LayerOfValue = 3
.Value = .FormatABC("date(now)")
.TimerInterval = 1000
endwith
```

## dBASE Plus

```
/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    Change = class::nativeObject_Change
endwith
*/
// Occurs when the layer's value is changed.
function nativeObject_Change(Layer)
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    oGauge.Layers.Item("sec").Value = oGauge.Value
    oGauge.Layers.Item("min").Value = oGauge.Value
    oGauge.Layers.Item("hour").Value = oGauge.Value
return

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    Timer = class::nativeObject_Timer
endwith
*/
// Occurs when the interval elapses.
function nativeObject_Timer(TickCount)
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    oGauge.Value = oGauge.FormatABC("value + 1/24/60/60",oGauge.Value)
return

local oGauge,var_Layer,var_Layer1,var_Layer2,var_Layer3

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
oGauge.Template = [DefaultLayer(185) = 2] // oGauge.DefaultLayer(185) = 2
```

```

oGauge.Layers.Count = 4
var_Layer = oGauge.Layers.Item(0)
    var_Layer.Background.Picture.Name = "vista_clock.png"
var_Layer1 = oGauge.Layers.Item(1)
    var_Layer1.Position = 3
    var_Layer1.Key = "sec"
    var_Layer1.OnDrag = 2
    var_Layer1.Selectable = false
    var_Layer1.Background.Picture.Name = "second-hand.png"
    var_Layer1.ValueToRotateAngle = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 60 )) -
floor(=:2) ) * 360"
    var_Layer1.RotateAngleToValue = "value / 360 / 24 / 60"
var_Layer2 = oGauge.Layers.Item(2)
    var_Layer2.Position = 2
    var_Layer2.Key = "min"
    var_Layer2.OnDrag = 2
    var_Layer2.Selectable = false
    var_Layer2.Background.Picture.Name = "Minute.png"
    var_Layer2.ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 360"
    var_Layer2.RotateAngleToValue = "value / 360 / 24 / 60"
var_Layer3 = oGauge.Layers.Item(3)
    var_Layer3.Position = 1
    var_Layer3.Key = "hour"
    var_Layer3.OnDrag = 2
    var_Layer3.Selectable = false
    var_Layer3.Background.Picture.Name = "Hour.png"
    var_Layer3.ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
    var_Layer3.RotateAngleToValue = "value / 360 * 0.5"
oGauge.LayerOfValue = 3
oGauge.Value = oGauge.FormatABC("date('now')")
oGauge.TimerInterval = 1000

```

*' Occurs when the layer's value is changed.*

```
function Change as v (Layer as N)
    oGauge = topparent:CONTROL_ACTIVEX1.activex
    oGauge.Layers.Item("sec").Value = oGauge.Value
    oGauge.Layers.Item("min").Value = oGauge.Value
    oGauge.Layers.Item("hour").Value = oGauge.Value
end function
```

*' Occurs when the interval elapses.*

```
function Timer as v (TickCount as N)
    oGauge = topparent:CONTROL_ACTIVEX1.activex
    oGauge.Value = oGauge.FormatABC("value + 1/24/60/60",oGauge.Value)
end function
```

```
Dim oGauge as P
Dim var_Layer as P
Dim var_Layer1 as P
Dim var_Layer2 as P
Dim var_Layer3 as P
```

```
oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
oGauge.Template = "DefaultLayer(185) = 2" // oGauge.DefaultLayer(185) = 2
oGauge.Layers.Count = 4
var_Layer = oGauge.Layers.Item(0)
    var_Layer.Background.Picture.Name = "vista_clock.png"
var_Layer1 = oGauge.Layers.Item(1)
    var_Layer1.Position = 3
    var_Layer1.Key = "sec"
    var_Layer1.OnDrag = 2
    var_Layer1.Selectable = .f
    var_Layer1.Background.Picture.Name = "second-hand.png"
    var_Layer1.ValueToRotateAngle = "((2:=(((1:=((0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 60 )) -
floor(=:2) ) * 360"
    var_Layer1.RotateAngleToValue = "value / 360 / 24 / 60"
```

```

var_Layer2 = oGauge.Layers.Item(2)
    var_Layer2.Position = 2
    var_Layer2.Key = "min"
    var_Layer2.OnDrag = 2
    var_Layer2.Selectable = .f
    var_Layer2.Background.Picture.Name = "Minute.png"
    var_Layer2.ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 360"
    var_Layer2.RotateAngleToValue = "value / 360 / 24 / 60"
var_Layer3 = oGauge.Layers.Item(3)
    var_Layer3.Position = 1
    var_Layer3.Key = "hour"
    var_Layer3.OnDrag = 2
    var_Layer3.Selectable = .f
    var_Layer3.Background.Picture.Name = "Hour.png"
    var_Layer3.ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
    var_Layer3.RotateAngleToValue = "value / 360 * 0.5"
oGauge.LayerOfValue = 3
oGauge.Value = oGauge.FormatABC("date(`now`)")
oGauge.TimerInterval = 1000

```

## Visual Objects

```

METHOD OCX_Exontrol1Change(Layer) CLASS MainDialog
    // Change event - Occurs when the layer's value is changed.
    oDCOCX_Exontrol1:Layers:[Item,"sec"]:Value := oDCOCX_Exontrol1:Value
    oDCOCX_Exontrol1:Layers:[Item,"min"]:Value := oDCOCX_Exontrol1:Value
    oDCOCX_Exontrol1:Layers:[Item,"hour"]:Value := oDCOCX_Exontrol1:Value
RETURN NIL

METHOD OCX_Exontrol1Timer(TickCount) CLASS MainDialog
    // Timer event - Occurs when the interval elapses.
    oDCOCX_Exontrol1:Value := oDCOCX_Exontrol1:FormatABC("value +
1/24/60/60",oDCOCX_Exontrol1:Value,nil,nil)
RETURN NIL

```

```
local var_Layer,var_Layer1,var_Layer2,var_Layer3 as ILayer
```

```
oDCOCX_Exontrol1:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
```

```
oDCOCX_Exontrol1:[DefaultLayer,exDefLayerRotateType] := 2
```

```
oDCOCX_Exontrol1:Layers:Count := 4
```

```
var_Layer := oDCOCX_Exontrol1:Layers:[Item,0]
```

```
var_Layer:Background:Picture:Name := "vista_clock.png"
```

```
var_Layer1 := oDCOCX_Exontrol1:Layers:[Item,1]
```

```
var_Layer1:Position := 3
```

```
var_Layer1:Key := "sec"
```

```
var_Layer1:OnDrag := exDoRotate
```

```
var_Layer1:Selectable := false
```

```
var_Layer1:Background:Picture:Name := "second-hand.png"
```

```
var_Layer1:ValueToRotateAngle := "((2:=(((1:= ( ( 0:=(value < 0 ? floor(value) + 1 -  
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 60 )) -  
floor(=:2)) * 360"
```

```
var_Layer1:RotateAngleToValue := "value / 360 / 24 / 60"
```

```
var_Layer2 := oDCOCX_Exontrol1:Layers:[Item,2]
```

```
var_Layer2:Position := 2
```

```
var_Layer2:Key := "min"
```

```
var_Layer2:OnDrag := exDoRotate
```

```
var_Layer2:Selectable := false
```

```
var_Layer2:Background:Picture:Name := "Minute.png"
```

```
var_Layer2:ValueToRotateAngle := "((1:= ( ( 0:=(value < 0 ? floor(value) + 1 - value :  
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 360"
```

```
var_Layer2:RotateAngleToValue := "value / 360 / 24 / 60"
```

```
var_Layer3 := oDCOCX_Exontrol1:Layers:[Item,3]
```

```
var_Layer3:Position := 1
```

```
var_Layer3:Key := "hour"
```

```
var_Layer3:OnDrag := exDoRotate
```

```
var_Layer3:Selectable := false
```

```
var_Layer3:Background:Picture:Name := "Hour.png"
```

```
var_Layer3:ValueToRotateAngle := "2 * 360 * ( 0:=(value < 0 ? floor(value) + 1 -  
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
```

```
var_Layer3:RotateAngleToValue := "value / 360 * 0.5"
```



```
oDCOCX_Exontrol1:LayerOfValue := 3
oDCOCX_Exontrol1:Value := oDCOCX_Exontrol1:FormatABC("date(now)",nil,nil,nil)
oDCOCX_Exontrol1:TimerInterval := 1000
```

## PowerBuilder

```
/*begin event Change(long Layer) - Occurs when the layer's value is changed.*/
/*
    oGauge = ole_1.Object
    oGauge.Layers.Item("sec").Value = oGauge.Value
    oGauge.Layers.Item("min").Value = oGauge.Value
    oGauge.Layers.Item("hour").Value = oGauge.Value
*/
/*end event Change*/

/*begin event Timer(long TickCount) - Occurs when the interval elapses.*/
/*
    oGauge = ole_1.Object
    oGauge.Value = oGauge.FormatABC("value + 1/24/60/60",oGauge.Value)
*/
/*end event Timer*/
```

```
OleObject oGauge,var_Layer,var_Layer1,var_Layer2,var_Layer3
```

```
oGauge = ole_1.Object
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
oGauge.DefaultLayer(185,2)
oGauge.Layers.Count = 4
var_Layer = oGauge.Layers.Item(0)
    var_Layer.Background.Picture.Name = "vista_clock.png"
var_Layer1 = oGauge.Layers.Item(1)
    var_Layer1.Position = 3
    var_Layer1.Key = "sec"
    var_Layer1.OnDrag = 2
    var_Layer1.Selectable = false
```

```

var_Layer1.Background.Picture.Name = "second-hand.png"
var_Layer1.ValueToRotateAngle = "((2:=(((1:=( ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 60 )) -
floor(=:2) ) * 360"
var_Layer1.RotateAngleToValue = "value / 360 / 24 / 60"
var_Layer2 = oGauge.Layers.Item(2)
var_Layer2.Position = 2
var_Layer2.Key = "min"
var_Layer2.OnDrag = 2
var_Layer2.Selectable = false
var_Layer2.Background.Picture.Name = "Minute.png"
var_Layer2.ValueToRotateAngle = "((1:=( ( (0:=(value < 0 ? floor(value) + 1 - value :
value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 360"
var_Layer2.RotateAngleToValue = "value / 360 / 24 / 60"
var_Layer3 = oGauge.Layers.Item(3)
var_Layer3.Position = 1
var_Layer3.Key = "hour"
var_Layer3.OnDrag = 2
var_Layer3.Selectable = false
var_Layer3.Background.Picture.Name = "Hour.png"
var_Layer3.ValueToRotateAngle = "2 * 360 * ( (0:=(value < 0 ? floor(value) + 1 -
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"
var_Layer3.RotateAngleToValue = "value / 360 * 0.5"
oGauge.LayerOfValue = 3
oGauge.Value = oGauge.FormatABC("date(`now`)")
oGauge.TimerInterval = 1000

```

## Visual DataFlex

```

// Occurs when the layer's value is changed.
Procedure OnComChange Integer IILayer
    Forward Send OnComChange IILayer
    Variant voLayers
    Get ComLayers to voLayers
    Handle hoLayers
    Get Create (RefClass(cComLayers)) to hoLayers

```

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComItem of hoLayers "sec" to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Variant v

Get ComValue to v

Set ComValue of hoLayer to v

Send Destroy to hoLayer

Send Destroy to hoLayers

Variant voLayers1

Get ComLayers to voLayers1

Handle hoLayers1

Get Create (RefClass(cComLayers)) to hoLayers1

Set pvComObject of hoLayers1 to voLayers1

Variant voLayer1

Get ComItem of hoLayers1 "min" to voLayer1

Handle hoLayer1

Get Create (RefClass(cComLayer)) to hoLayer1

Set pvComObject of hoLayer1 to voLayer1

Variant v1

Get ComValue to v1

Set ComValue of hoLayer1 to v1

Send Destroy to hoLayer1

Send Destroy to hoLayers1

Variant voLayers2

Get ComLayers to voLayers2

Handle hoLayers2

Get Create (RefClass(cComLayers)) to hoLayers2

Set pvComObject of hoLayers2 to voLayers2

Variant voLayer2

Get ComItem of hoLayers2 "hour" to voLayer2

Handle hoLayer2

Get Create (RefClass(cComLayer)) to hoLayer2

Set pvComObject of hoLayer2 to voLayer2

Variant v2

```
    Get ComValue to v2
    Set ComValue of hoLayer2 to v2
    Send Destroy to hoLayer2
    Send Destroy to hoLayers2
End_Procedure
```

*// Occurs when the interval elapses.*

```
Procedure OnComTimer Integer IITickCount
    Forward Send OnComTimer IITickCount
    Set ComValue to (ComFormatABC(Self,"value + 1/24/60/60",
(ComValue(Self)),Nothing,Nothing))
End_Procedure
```

```
Procedure OnCreate
    Forward Send OnCreate
    Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"
    Set ComDefaultLayer OLExDefLayerRotateType to 2
    Variant voLayers3
    Get ComLayers to voLayers3
    Handle hoLayers3
    Get Create (RefClass(cComLayers)) to hoLayers3
    Set pvComObject of hoLayers3 to voLayers3
        Set ComCount of hoLayers3 to 4
    Send Destroy to hoLayers3
    Variant voLayers4
    Get ComLayers to voLayers4
    Handle hoLayers4
    Get Create (RefClass(cComLayers)) to hoLayers4
    Set pvComObject of hoLayers4 to voLayers4
        Variant voLayer3
        Get ComItem of hoLayers4 0 to voLayer3
        Handle hoLayer3
        Get Create (RefClass(cComLayer)) to hoLayer3
        Set pvComObject of hoLayer3 to voLayer3
            Variant voBackground
            Get ComBackground of hoLayer3 to voBackground
```

Handle hoBackground  
Get Create (RefClass(cComBackground)) to hoBackground  
Set pvComObject of hoBackground to voBackground  
Variant voPicture  
Get ComPicture of hoBackground to voPicture  
Handle hoPicture  
Get Create (RefClass(cComPicture)) to hoPicture  
Set pvComObject of hoPicture to voPicture  
Set ComName of hoPicture to "vista\_clock.png"  
Send Destroy to hoPicture  
Send Destroy to hoBackground  
Send Destroy to hoLayer3  
Send Destroy to hoLayers4  
Variant voLayers5  
Get ComLayers to voLayers5  
Handle hoLayers5  
Get Create (RefClass(cComLayers)) to hoLayers5  
Set pvComObject of hoLayers5 to voLayers5  
Variant voLayer4  
Get ComItem of hoLayers5 1 to voLayer4  
Handle hoLayer4  
Get Create (RefClass(cComLayer)) to hoLayer4  
Set pvComObject of hoLayer4 to voLayer4  
Set ComPosition of hoLayer4 to 3  
Set ComKey of hoLayer4 to "sec"  
Set ComOnDrag of hoLayer4 to OLEexDoRotate  
Set ComSelectable of hoLayer4 to False  
Variant voBackground1  
Get ComBackground of hoLayer4 to voBackground1  
Handle hoBackground1  
Get Create (RefClass(cComBackground)) to hoBackground1  
Set pvComObject of hoBackground1 to voBackground1  
Variant voPicture1  
Get ComPicture of hoBackground1 to voPicture1  
Handle hoPicture1  
Get Create (RefClass(cComPicture)) to hoPicture1  
Set pvComObject of hoPicture1 to voPicture1

Set ComName of hoPicture1 to "second-hand.png"

Send Destroy to hoPicture1

Send Destroy to hoBackground1

Set ComValueToRotateAngle of hoLayer4 to " $((2:=(((1:=((0:=(value < 0 ? floor(value) + 1 - value : value - floor(value)))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - floor(=:1)) * 60 )) - floor(=:2) ) * 360$ "

Set ComRotateAngleToValue of hoLayer4 to " $value / 360 / 24 / 60$ "

Send Destroy to hoLayer4

Send Destroy to hoLayers5

Variant voLayers6

Get ComLayers to voLayers6

Handle hoLayers6

Get Create (RefClass(cComLayers)) to hoLayers6

Set pvComObject of hoLayers6 to voLayers6

Variant voLayer5

Get ComItem of hoLayers6 2 to voLayer5

Handle hoLayer5

Get Create (RefClass(cComLayer)) to hoLayer5

Set pvComObject of hoLayer5 to voLayer5

Set ComPosition of hoLayer5 to 2

Set ComKey of hoLayer5 to "min"

Set ComOnDrag of hoLayer5 to OLEexDoRotate

Set ComSelectable of hoLayer5 to False

Variant voBackground2

Get ComBackground of hoLayer5 to voBackground2

Handle hoBackground2

Get Create (RefClass(cComBackground)) to hoBackground2

Set pvComObject of hoBackground2 to voBackground2

Variant voPicture2

Get ComPicture of hoBackground2 to voPicture2

Handle hoPicture2

Get Create (RefClass(cComPicture)) to hoPicture2

Set pvComObject of hoPicture2 to voPicture2

Set ComName of hoPicture2 to "Minute.png"

Send Destroy to hoPicture2

Send Destroy to hoBackground2

Set ComValueToRotateAngle of hoLayer5 to " $((1:=((0:=(value < 0 ?$

$\text{floor}(\text{value}) + 1 - \text{value} : \text{value} - \text{floor}(\text{value})) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) * 24 )) - \text{floor}(=:1)) * 360"$

Set ComRotateAngleToValue of hoLayer5 to " $\text{value} / 360 / 24 / 60$ "

Send Destroy to hoLayer5

Send Destroy to hoLayers6

Variant voLayers7

Get ComLayers to voLayers7

Handle hoLayers7

Get Create (RefClass(cComLayers)) to hoLayers7

Set pvComObject of hoLayers7 to voLayers7

Variant voLayer6

Get ComItem of hoLayers7 3 to voLayer6

Handle hoLayer6

Get Create (RefClass(cComLayer)) to hoLayer6

Set pvComObject of hoLayer6 to voLayer6

Set ComPosition of hoLayer6 to 1

Set ComKey of hoLayer6 to "**hour**"

Set ComOnDrag of hoLayer6 to OLEexDoRotate

Set ComSelectable of hoLayer6 to False

Variant voBackground3

Get ComBackground of hoLayer6 to voBackground3

Handle hoBackground3

Get Create (RefClass(cComBackground)) to hoBackground3

Set pvComObject of hoBackground3 to voBackground3

Variant voPicture3

Get ComPicture of hoBackground3 to voPicture3

Handle hoPicture3

Get Create (RefClass(cComPicture)) to hoPicture3

Set pvComObject of hoPicture3 to voPicture3

Set ComName of hoPicture3 to "**Hour.png**"

Send Destroy to hoPicture3

Send Destroy to hoBackground3

Set ComValueToRotateAngle of hoLayer6 to " $2 * 360 * (0:= (\text{value} < 0 ?$

$\text{floor}(\text{value}) + 1 - \text{value} : \text{value} - \text{floor}(\text{value})) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"$

Set ComRotateAngleToValue of hoLayer6 to " $\text{value} / 360 * 0.5$ "

Send Destroy to hoLayer6

Send Destroy to hoLayers7

```
Set ComLayerOfValue to 3
Set ComValue to (ComFormatABC(Self,"date(now)",Nothing,Nothing,Nothing))
Set ComTimerInterval to 1000
End_Procedure
```

## XBase++

```
PROCEDURE OnChange(oGauge,Layer)
  oGauge:Layers:Item("sec"):Value := oGauge:Value()
  oGauge:Layers:Item("min"):Value := oGauge:Value()
  oGauge:Layers:Item("hour"):Value := oGauge:Value()
RETURN

PROCEDURE OnTimer(oGauge,TickCount)
  oGauge:Value := oGauge:FormatABC("value + 1/24/60/60",oGauge:Value())
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oGauge
  LOCAL oLayer,oLayer1,oLayer2,oLayer3

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oGauge := XbpActiveXControl():new( oForm:drawingArea )
  oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
  oGauge:create(,, {10,60},{610,370} )

  oGauge:Change := {|| Layer| OnChange(oGauge,Layer)} /*Occurs when the layer's
```



*value is changed.\**

oGauge:Timer := {|TickCount| OnTimer(oGauge,TickCount)} */\*Occurs when the interval elapses.\**

oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Clock"

oGauge:SetProperty("DefaultLayer",185*/\*exDefLayerRotateType\*/*,2)

oGauge:Layers():Count := 4

oLayer := oGauge:Layers:Item(0)

oLayer:Background():Picture():Name := "vista\_clock.png"

oLayer1 := oGauge:Layers:Item(1)

oLayer1:Position := 3

oLayer1:Key := "sec"

oLayer1:OnDrag := 2*/\*exDoRotate\*/*

oLayer1:Selectable := .F.

oLayer1:Background():Picture():Name := "second-hand.png"

oLayer1:ValueToRotateAngle := "((2:=(((1:= ( ( 0:=(value < 0 ? floor(value) + 1 -  
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) \* 24 )) - floor(=:1)) \* 60 )) -  
floor(=:2) ) \* 360"

oLayer1:RotateAngleToValue := "value / 360 / 24 / 60"

oLayer2 := oGauge:Layers:Item(2)

oLayer2:Position := 2

oLayer2:Key := "min"

oLayer2:OnDrag := 2*/\*exDoRotate\*/*

oLayer2:Selectable := .F.

oLayer2:Background():Picture():Name := "Minute.png"

oLayer2:ValueToRotateAngle := "((1:= ( ( 0:=(value < 0 ? floor(value) + 1 -  
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) \* 24 )) - floor(=:1)) \* 360"

oLayer2:RotateAngleToValue := "value / 360 / 24 / 60"

oLayer3 := oGauge:Layers:Item(3)

oLayer3:Position := 1

oLayer3:Key := "hour"

oLayer3:OnDrag := 2*/\*exDoRotate\*/*

oLayer3:Selectable := .F.

oLayer3:Background():Picture():Name := "Hour.png"

oLayer3:ValueToRotateAngle := "2 \* 360 \* ( ( 0:=(value < 0 ? floor(value) + 1 -  
value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )"

```
oLayer3:RotateAngleToValue := "value / 360 * 0.5"  
oGauge:LayerOfValue := 3  
oGauge:Value := oGauge:FormatABC("date(`now`)")  
oGauge:TimerInterval := 1000  
  
oForm:Show()  
DO WHILE nEvent != xbeP_Quit  
    nEvent := AppEvent( @mp1, @mp2, @oXbp )  
    oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

# property Gauge.ToolTipDelay as Long

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

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

By default, the ToolTipDelay property is 500, which indicates that the tooltip is shown after 0.5 seconds. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [ShowToolTip](#) method to display a custom tooltip. The [ToolTip](#) / [ToolTipTitle](#) property indicates the layer's tooltip. The [LayerFromPoint](#) property returns the index of the layer from the cursor. Use the [ToolTipWidth](#) property to specify the width of the tooltip window Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

# property Gauge.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

Type	Description
IFontDisp	A Font object to be used by the control's tooltip.

Use the ToolTipFont property to change the tooltip's font. Use the [ShowToolTip](#) method to display a custom tooltip. The [ToolTip](#) / [ToolTipTitle](#) property indicates the layer's tooltip. The [LayerFromPoint](#) property returns the index of the layer from the cursor. Use the [ToolTipWidth](#) property to specify the width of the tooltip window Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

# property Gauge.ToolTipPopDelay as Long

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

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

By default, the ToolTipPopDelay property is 5000, which indicates that the tooltip remains visible for 5 seconds, while the cursor is not moved. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [ShowToolTip](#) method to display a custom tooltip. The [ToolTip](#) / [ToolTipTitle](#) property indicates the layer's tooltip. The [LayerFromPoint](#) property returns the index of the layer from the cursor. Use the [ToolTipWidth](#) property to specify the width of the tooltip window Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

# property Gauge.ToolTipWidth as Long

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

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

By default, the ToolTipWidth property is 196 pixels. Use the ToolTipWidth property to specify the width of the tooltip window. Use the [ShowToolTip](#) method to display a custom tooltip. The [ToolTip](#) / [ToolTipTitle](#) property indicates the layer's tooltip. The [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. The [LayerFromPoint](#) property returns the index of the layer from the cursor. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

# property Gauge.ToTemplate ([DefaultTemplate as Variant]) as String

Generates the control's template.

Type	Description
DefaultTemplate as Variant	A String expression that indicates the default format used to define the control's template at runtime, or a string expression that indicates the path to the file being used to define the default template ( like c:\temp\templ.bin ). If it is missing ( by default ), the control's uses the default implementation ( listed bellow ) to define the control's template, at runtime. Each line in the DefaultTemplate parameter, defines a property or an instruction to generate the template.
String	A String expression that indicates the control's template.

Use the ToTemplate property to save the control's content to a template string. The ToTemplate property saves the control's properties based on the default template. Use the ToTemplate property to copy the control's content to another instance. The ToTemplate property can save pictures, icons, binary arrays, objects, collections, and so on based on the DefaultTemplate parameter.

The DefaultTemplate parameter indicates the format of the template being used to generate the control's template at runtime. If the DefaultTemplate parameter is missing, the control's uses its default template listed bellow. The DefaultTemplate parameter defines the list of properties and instructions that generates the control's template. Remove the properties and objects, in the default template, that you don't need in the generated template script. Use the [Template](#) property to apply the template to the control. Use the Template property to execute code by passing instructions as a string ( template string ). The Template script is composed by lines of instructions. Instructions are separated by "\n\r" ( newline) characters. The Template format contains a list of instructions that loads data and change properties for the objects in the control. Use the [AllowCopyTemplate](#) property to copy the control's content to the clipboard, in template format, using the the Shift + Ctrl + Alt + Insert sequence.

# property Gauge.TransparentColorFrom as Color

Specifies the transparent color for all pictures in all layers, to define transparency part (from).

Type	Description
Color	A Color expression that defines the transparent color to be applied on all pictures on any layer.

By default, the TransparentColorFrom property is RGB( 255, 255, 255 ) and TransparentColorTo property is -1, which indicates that pixels of white colors are transparent. The TransparentColorFrom property defines the transparent color for all pictures on any layer, that has the [TransparentColorFrom](#) property on -1 ( by default ). The [TransparentColorTo](#) defines the second transparent color, to define transparent pixels between a range of colors. The [Opaque](#) property indicates if the picture is shown as opaque or transparent.

The TransparentColorFrom / TransparentColorTo properties have effect it:

- [Opaque](#) property is False ( by default )
- picture's attribute does **not** include the PICTURE\_TRANSPARENT flag ( for instance a PNG picture with transparency, includes the PICTURE\_TRANSPARENT flag )
- TransparentColorFrom / TransparentColorTo properties points to valid colors ( different than -1 value ). For instance, if one property is defined and the other is -1, the first one defines the transparent pixels, while if both are specified and points to value different than -1, any pixel between them is considered as transparent.

If The TransparentColorFrom / TransparentColorTo properties have effect, any picture where these apply defines the pixels as:

- any pixel with a color between TransparentColorFrom and TransparentColorTo is defined as transparent
- any other pixel that's not transparent is opaque.

If using the **PNG** format, the control handles automatically its transparency / alpha blending ( if saved with transparency ), unless the [Opaque](#) property is True, so in this case, any TransparentColorFrom or TransparentColorTo property has no effect.

For any other picture type, you can use any of the following to define the transparent region of the picture:

- TransparentColorFrom, specifies the transparent color to define transparency part of the current picture (to).
- TransparentColorTo, specifies the transparent color to define transparency part of the



current picture (to).

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

# property Gauge.TransparentColorTo as Color

Specifies the transparent color for all pictures in all layers, to define transparency part (to).

Type	Description
Color	A Color expression that defines the second transparent color to be applied on all pictures on any layer.

By default, the [TransparentColorFrom](#) property is RGB( 255, 255, 255 ) and TransparentColorTo property is -1, which indicates that pixels of white colors are transparent. The TransparentColorFrom property defines the transparent color for all pictures on any layer, that has the [TransparentColorFrom](#) property on -1 ( by default ). The TransparentColorTo defines the second transparent color, to define transparent pixels between a range of colors. The [Opaque](#) property indicates if the picture is shown as opaque or transparent.

The TransparentColorFrom / TransparentColorTo properties have effect it:

- [Opaque](#) property is False ( by default )
- picture's attribute does **not** include the PICTURE\_TRANSPARENT flag ( for instance a PNG picture with transparency, includes the PICTURE\_TRANSPARENT flag )
- TransparentColorFrom / TransparentColorTo properties points to valid colors ( different than -1 value ). For instance, if one property is defined and the other is -1, the first one defines the transparent pixels, while if both are specified and points to value different than -1, any pixel between them is considered as transparent.

If The TransparentColorFrom / TransparentColorTo properties have effect, any picture where these apply defines the pixels as:

- any pixel with a color between TransparentColorFrom and TransparentColorTo is defined as transparent
- any other pixel that's not transparent is opaque.

If using the **PNG** format, the control handles automatically its transparency / alpha blending ( if saved with transparency ), unless the [Opaque](#) property is True, so in this case, any TransparentColorFrom or TransparentColorTo property has no effect.

For any other picture type, you can use any of the following to define the transparent region of the picture:

- TransparentColorFrom, specifies the transparent color to define transparency part of the current picture (to).
- TransparentColorTo, specifies the transparent color to define transparency part of the current picture (to).

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

# Property Gauge.Value as Variant

Specifies the control's value.

Type	Description
Variant	A VARIANT expression that specifies the value associated with the control.

By default, the Value property is empty. The layer's [Value](#) could indicate its offset or its rotation angle, based on the [OnDrag](#) property. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). Use the [Value](#) property of the Clip object to associate a value with the layer's clipping region. Each layer can associate a value with it, while the control's Value property can be associated through the [LayerOfValue](#) property with the value of one of the layers within the control.

For instance:

- the control displays a clock, the value could be the current-time
- the control shows a switch, so the value could indicate the state of the switch
- the control shows a thermometer, so the value could be the current temperature
- the control displays a gauge, so the value could be the value on the gauge pointed by the needle

The [Change](#) event occurs when the layer's Value property is changed. During the Change event, you can change values of other layers as well. For instance, if the second-hand of the clock is rotated, you can rotate the hour and the minute-hands of the clock as well. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. *You can call `DragInfo.Debug = -1` during the [DragStart](#) event to display debugging information like current movement, rotation angle when drag operation is performed.*

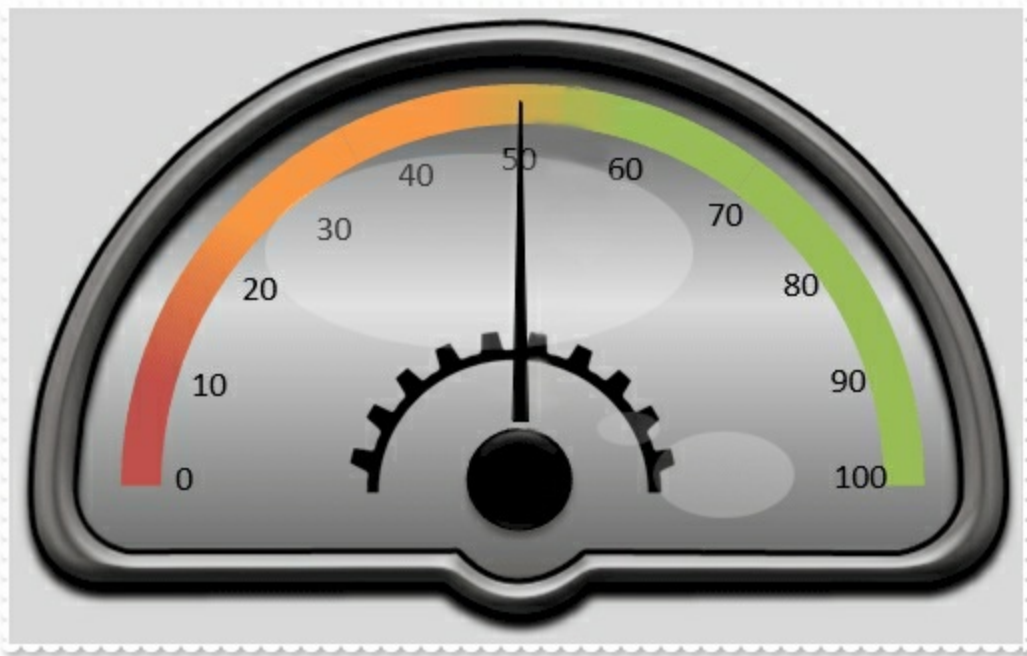
The Value property indicates the **value** keyword in the following properties:

- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the [ValueToOffsetX](#) expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**. The [RotateAngleToValue](#) converts the current rotation angle to a value.

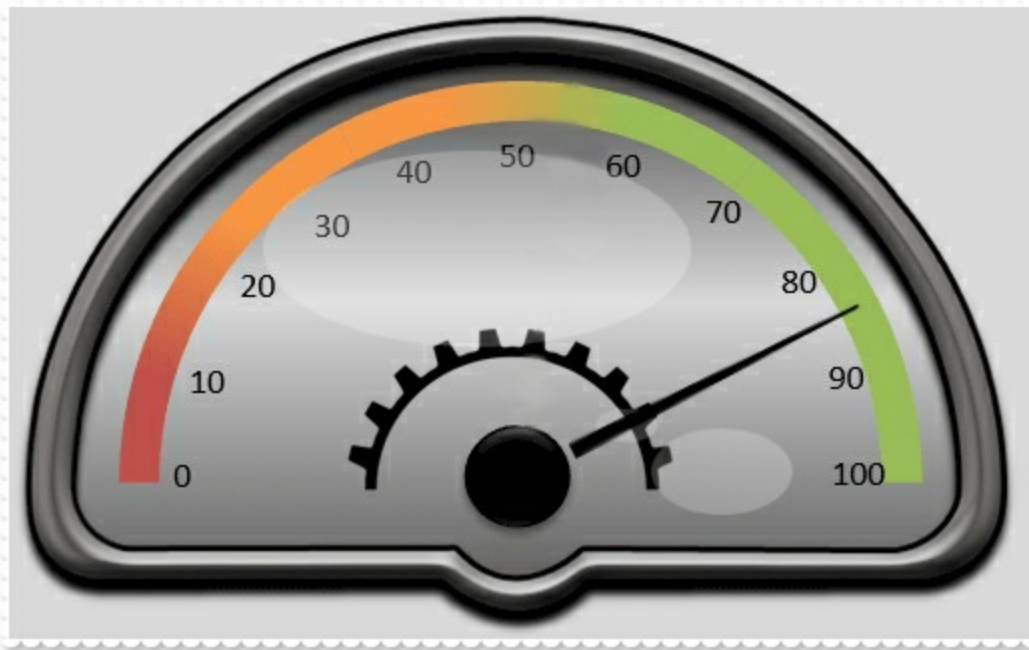
The Value property works in association with the layer's [OnDrag](#) property like follows:

- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetX](#) property indicates the layer's [OffsetX](#) property.
- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetY](#) property indicates the layer's [OffsetY](#) property.
- If the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**, evaluating the [ValueToRotateAngle](#) property indicates the layer's [RotateAngle](#) property.

For instance, having the gauge from the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Guage folder, which includes the background and the needle pictures:



we need to define the value of the needle to be between 0 and 100, so if we call Value property on 85 we should get something like:



In conclusion, what we need to do is:

- defines the "needle" layer as rotate able, using the [OnDrag](#) property
- converts the value of 0-100, to a rotation angle, using the [ValueToRotateAngle](#) property
- converts the rotation angle from 0-360 to the value, using the [RotateAngleToValue](#) property
- limits the rotation angle, using the [RotateAngleValid](#) property

The following samples shows how you can do that:

### VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .DefaultLayer(185) = 2
    .BackColor = RGB(217,217,217)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    With .Layers.Add("background")
        .Background.Picture.Name = "Guage_Background.png"
        .RotateCenterY = "Iheight/2 + 78"
    End With
    With .Layers.Add("needle")
        .Background.Picture.Name = "Guage_Needle.png"
```

```

.OnDrag = 2
.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))"
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
End With
.Value = 85
.EndUpdate
End With

```

## VB6

```

With Gauge1
.BeginUpdate
.DefaultLayer(exDefLayerRotateType) = 2
.BackColor = RGB(217,217,217)
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
With .Layers.Add("background")
.Background.Picture.Name = "Guage_Background.png"
.RotateCenterY = "lheight/2 + 78"
End With
With .Layers.Add("needle")
.Background.Picture.Name = "Guage_Needle.png"
.OnDrag = exDoRotate
.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))"
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
End With
.Value = 85
.EndUpdate
End With

```

## VB.NET

```

With Exgauge1
.BeginUpdate()

```

```

.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateTy

    .BackColor = Color.FromArgb(217,217,217)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    With .Layers.Add("background")
        .Background.Picture.Name = "Guage_Background.png"
        .RotateCenterY = "lheight/2 + 78"
    End With
    With .Layers.Add("needle")
        .Background.Picture.Name = "Guage_Needle.png"
        .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
        .RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))"
        .RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
        .ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
    End With
    .Value = 85
    .EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,2)
    .BackColor = RGB(217,217,217)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    With .Layers.Add("background")
        .Background.Picture.Name = "Guage_Background.png"
        .RotateCenterY = "lheight/2 + 78"
    End With
    With .Layers.Add("needle")
        .Background.Picture.Name = "Guage_Needle.png"
        .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
    End With
End With

```



```

.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <270 ? 270 :
value ))"
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
End With
.Value = 85
.EndUpdate()
End With

```

## C++

```

/*
Copy and paste the following directives to your header file as
it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
Library'

#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
>GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerRotateType,long(2));
spGauge1->PutBackColor(RGB(217,217,217));
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage");
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->Add("background");
var_Layer->GetBackground()->GetPicture()->PutName("Guage_Background.png");
var_Layer->PutRotateCenterY(L"height/2 + 78");
EXGAUGELib::ILayerPtr var_Layer1 = spGauge1->GetLayers()->Add("needle");
var_Layer1->GetBackground()->GetPicture()->PutName("Guage_Needle.png");
var_Layer1->PutOnDrag(EXGAUGELib::exDoRotate);
var_Layer1->PutRotateAngleValid(L"value < 90 ? value : (value < 180 ? 90 : ( value
< 270 ? 270 : value ))");
var_Layer1->PutRotateAngleToValue(L"value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50");
var_Layer1->PutValueToRotateAngle(L"value < 50 ? (270 + value/50*90) : (value -

```

```
50)/50 * 90");
spGauge1->PutValue(long(85));
spGauge1->EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();
Gauge1-
>DefaultLayer[ExgaugeLib_tlb::DefaultLayerPropertyEnum::exDefLayerRotateType] =
TVariant(2);
Gauge1->BackColor = RGB(217,217,217);
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
ExgaugeLib_tlb::ILayerPtr var_Layer = Gauge1->Layers-
>Add(TVariant("background"));
    var_Layer->Background->Picture->set_Name(TVariant("Guage_Background.png"));
    var_Layer->RotateCenterY = L"height/2 + 78";
ExgaugeLib_tlb::ILayerPtr var_Layer1 = Gauge1->Layers->Add(TVariant("needle"));
    var_Layer1->Background->Picture->set_Name(TVariant("Guage_Needle.png"));
    var_Layer1->OnDrag = ExgaugeLib_tlb::OnDragLayerEnum::exDoRotate;
    var_Layer1->RotateAngleValid = L"value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1->RotateAngleToValue = L"value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1->ValueToRotateAngle = L"value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
Gauge1->set_Value(TVariant(85));
Gauge1->EndUpdate();
```

## C#

```
exgauge1.BeginUpdate();
exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayer

exgauge1.BackColor = Color.FromArgb(217,217,217);
exgauge1.PicturesPath = "C:\\Program
```

```

Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers.Add("background");
    var_Layer.Background.Picture.Name = "Guage_Background.png";
    var_Layer.RotateCenterY = "Iheight/2 + 78";
exontrol.EXGAUGELib.Layer var_Layer1 = exgauge1.Layers.Add("needle");
    var_Layer1.Background.Picture.Name = "Guage_Needle.png";
    var_Layer1.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
exgauge1.Value = 85;
exgauge1.EndUpdate();

```

## JScript/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.DefaultLayer(185) = 2;
    Gauge1.BackColor = 14277081;
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
    var var_Layer = Gauge1.Layers.Add("background");
        var_Layer.Background.Picture.Name = "Guage_Background.png";
        var_Layer.RotateCenterY = "Iheight/2 + 78";
    var var_Layer1 = Gauge1.Layers.Add("needle");
        var_Layer1.Background.Picture.Name = "Guage_Needle.png";
        var_Layer1.OnDrag = 2;

```

```

    var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
    Gauge1.Value = 85;
    Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .DefaultLayer(185) = 2
        .BackColor = RGB(217,217,217)
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
        With .Layers.Add("background")
            .Background.Picture.Name = "Guage_Background.png"
            .RotateCenterY = "lheight/2 + 78"
        End With
        With .Layers.Add("needle")
            .Background.Picture.Name = "Guage_Needle.png"
            .OnDrag = 2
            .RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ?
270 : value ))"
            .RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 +

```

```

50"
    .ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 *
90"
    End With
    .Value = 85
    .EndUpdate
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotate

axGauge1.BackColor = Color.FromArgb(217,217,217);
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
EXGAUGELib.Layer var_Layer = axGauge1.Layers.Add("background");
    var_Layer.Background.Picture.Name = "Guage_Background.png";
    var_Layer.RotateCenterY = "lheight/2 + 78";
EXGAUGELib.Layer var_Layer1 = axGauge1.Layers.Add("needle");
    var_Layer1.Background.Picture.Name = "Guage_Needle.png";
    var_Layer1.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
axGauge1.Value = 85;
axGauge1.EndUpdate();

```

```

public void init()
{
    COM com_Background,com_Layer,com_Layer1,com_Picture;
    anytype var_Background,var_Layer,var_Layer1,var_Picture;
    ;

    super();

    exgauge1.BeginUpdate();

    exgauge1.DefaultLayer(185/*exDefLayerRotateType*/,COMVariant::createFromInt(2));
    exgauge1.BackColor(WinApi::RGB2int(217,217,217));
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage");
    var_Layer = COM::createFromObject(exgauge1.Layers()).Add("background");
    com_Layer = var_Layer;
    var_Background = COM::createFromObject(com_Layer.Background());
    com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
    com_Picture.Name("Guage_Background.png");
    com_Layer.RotateCenterY("lheight/2 + 78");
    var_Layer1 = COM::createFromObject(exgauge1.Layers()).Add("needle");
    com_Layer1 = var_Layer1;
    var_Background = COM::createFromObject(com_Layer1.Background());
    com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
    com_Picture.Name("Guage_Needle.png");
    com_Layer1.OnDrag(2/*exDoRotate*/);
    com_Layer1.RotateAngleValid("value < 90 ? value : (value < 180 ? 90 : (value <
270 ? 270 : value ))");
    com_Layer1.RotateAngleToValue("value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50");
    com_Layer1.ValueToRotateAngle("value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90");
    exgauge1.Value(COMVariant::createFromInt(85));

```

```
exgauge1.EndUpdate();  
}
```

## Delphi 8 (.NET only)

```
with AxGauge1 do  
begin  
  BeginUpdate();  
  
  set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType, TObj  
  
  BackColor := Color.FromArgb(217,217,217);  
  PicturesPath := 'C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage';  
  with Layers.Add('background') do  
  begin  
    Background.Picture.Name := 'Guage_Background.png';  
    RotateCenterY := 'lheight/2 + 78';  
  end;  
  with Layers.Add('needle') do  
  begin  
    Background.Picture.Name := 'Guage_Needle.png';  
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;  
    RotateAngleValid := 'value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :  
value ))';  
    RotateAngleToValue := 'value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50';  
    ValueToRotateAngle := 'value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90';  
  end;  
  Value := TObject(85);  
  EndUpdate();  
end
```

## Delphi (standard)

```
with Gauge1 do  
begin  
  BeginUpdate();  
  DefaultLayer[EXGAUGELib_TLB.exDefLayerRotateType] := OleVariant(2);
```

```

BackColor := RGB(217,217,217);
PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage';
with Layers.Add('background') do
begin
    Background.Picture.Name := 'Guage_Background.png';
    RotateCenterY := 'lheight/2 + 78';
end;
with Layers.Add('needle') do
begin
    Background.Picture.Name := 'Guage_Needle.png';
    OnDrag := EXGAUGELib_TLB.exDoRotate;
    RotateAngleValid := 'value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))';
    RotateAngleToValue := 'value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50';
    ValueToRotateAngle := 'value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90';
end;
Value := OleVariant(85);
EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.Object.DefaultLayer(185) = 2
.BackColor = RGB(217,217,217)
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
with .Layers.Add("background")
    .Background.Picture.Name = "Guage_Background.png"
    .RotateCenterY = "lheight/2 + 78"
endwith
with .Layers.Add("needle")
    .Background.Picture.Name = "Guage_Needle.png"
    .OnDrag = 2
    .RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :

```



```
value ))"
```

```
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
```

```
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
```

```
endwith
```

```
.Value = 85
```

```
.EndUpdate
```

```
endwith
```

## dBASE Plus

```
local oGauge,var_Layer,var_Layer1
```

```
oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
```

```
oGauge.BeginUpdate()
```

```
oGauge.Template = [DefaultLayer(185) = 2] // oGauge.DefaultLayer(185) = 2
```

```
oGauge.BackColor = 0xd9d9d9
```

```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
var_Layer = oGauge.Layers.Add("background")
```

```
var_Layer.Background.Picture.Name = "Guage_Background.png"
```

```
var_Layer.RotateCenterY = "Iheight/2 + 78"
```

```
var_Layer1 = oGauge.Layers.Add("needle")
```

```
var_Layer1.Background.Picture.Name = "Guage_Needle.png"
```

```
var_Layer1.OnDrag = 2
```

```
var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <  
270 ? 270 : value ))"
```

```
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :  
(value/90)*50 + 50"
```

```
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -  
50)/50 * 90"
```

```
oGauge.Value = 85
```

```
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P
```

```
Dim var_Layer as P
```

```
Dim var_Layer1 as P
```

```
oGauge = topparent:CONTROL_ACTIVEX1.activex
```

```
oGauge.BeginUpdate()
```

```
oGauge.Template = "DefaultLayer(185) = 2" // oGauge.DefaultLayer(185) = 2
```

```
oGauge.BackColor = 14277081
```

```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
var_Layer = oGauge.Layers.Add("background")
```

```
var_Layer.Background.Picture.Name = "Guage_Background.png"
```

```
var_Layer.RotateCenterY = "Iheight/2 + 78"
```

```
var_Layer1 = oGauge.Layers.Add("needle")
```

```
var_Layer1.Background.Picture.Name = "Guage_Needle.png"
```

```
var_Layer1.OnDrag = 2
```

```
var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <  
270 ? 270 : value ))"
```

```
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :  
(value/90)*50 + 50"
```

```
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -  
50)/50 * 90"
```

```
oGauge.Value = 85
```

```
oGauge.EndUpdate()
```

## Visual Objects

```
local var_Layer,var_Layer1 as ILayer
```

```
oDCOCX_Exontrol1:BeginUpdate()
```

```
oDCOCX_Exontrol1:[DefaultLayer,exDefLayerRotateType] := 2
```

```
oDCOCX_Exontrol1.BackColor := RGB(217,217,217)
```

```
oDCOCX_Exontrol1.PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
var_Layer := oDCOCX_Exontrol1.Layers.Add("background")
```

```
var_Layer:Background:Picture:Name := "Guage_Background.png"
```

```
var_Layer:RotateCenterY := "Iheight/2 + 78"
```

```
var_Layer1 := oDCOCX_Exontrol1.Layers.Add("needle")
```

```

var_Layer1:Background:Picture:Name := "Guage_Needle.png"
var_Layer1:OnDrag := exDoRotate
var_Layer1:RotateAngleValid := "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))"
var_Layer1:RotateAngleToValue := "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50"
var_Layer1:ValueToRotateAngle := "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90"
oDCOCX_Exontrol1:Value := 85
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge,var_Layer,var_Layer1

oGauge = ole_1.Object
oGauge.BeginUpdate()
oGauge.DefaultLayer(185,2)
oGauge.BackColor = RGB(217,217,217)
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
var_Layer = oGauge.Layers.Add("background")
var_Layer.Background.Picture.Name = "Guage_Background.png"
var_Layer.RotateCenterY = "lheight/2 + 78"
var_Layer1 = oGauge.Layers.Add("needle")
var_Layer1.Background.Picture.Name = "Guage_Needle.png"
var_Layer1.OnDrag = 2
var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))"
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50"
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90"
oGauge.Value = 85
oGauge.EndUpdate()

```

## Procedure OnCreate

Forward Send OnCreate

Send ComBeginUpdate

Set ComDefaultLayer OLExDefLayerRotateType to 2

Set ComBackColor to (RGB(217,217,217))

Set ComPicturesPath to "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComAdd of hoLayers "background" to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Variant voBackground

Get ComBackground of hoLayer to voBackground

Handle hoBackground

Get Create (RefClass(cComBackground)) to hoBackground

Set pvComObject of hoBackground to voBackground

Variant voPicture

Get ComPicture of hoBackground to voPicture

Handle hoPicture

Get Create (RefClass(cComPicture)) to hoPicture

Set pvComObject of hoPicture to voPicture

Set ComName of hoPicture to "Guage\_Background.png"

Send Destroy to hoPicture

Send Destroy to hoBackground

Set ComRotateCenterY of hoLayer to "lheight/2 + 78"

Send Destroy to hoLayer

Send Destroy to hoLayers

Variant voLayers1

Get ComLayers to voLayers1

```

Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer1
    Get ComAdd of hoLayers1 "needle" to voLayer1
    Handle hoLayer1
    Get Create (RefClass(cComLayer)) to hoLayer1
    Set pvComObject of hoLayer1 to voLayer1
        Variant voBackground1
        Get ComBackground of hoLayer1 to voBackground1
        Handle hoBackground1
        Get Create (RefClass(cComBackground)) to hoBackground1
        Set pvComObject of hoBackground1 to voBackground1
            Variant voPicture1
            Get ComPicture of hoBackground1 to voPicture1
            Handle hoPicture1
            Get Create (RefClass(cComPicture)) to hoPicture1
            Set pvComObject of hoPicture1 to voPicture1
                Set ComName of hoPicture1 to "Guage_Needle.png"
            Send Destroy to hoPicture1
        Send Destroy to hoBackground1
    Set ComOnDrag of hoLayer1 to OLEexDoRotate
    Set ComRotateAngleValid of hoLayer1 to "value < 90 ? value : (value < 180 ?
90 : ( value < 270 ? 270 : value ))"
    Set ComRotateAngleToValue of hoLayer1 to "value >= 270 ? (value -
270)/90*50 : (value/90)*50 + 50"
    Set ComValueToRotateAngle of hoLayer1 to "value < 50 ? (270 + value/50*90)
: (value - 50)/50 * 90"
    Send Destroy to hoLayer1
Send Destroy to hoLayers1
Set ComValue to 85
Send ComEndUpdate
End_Procedure

```

**XBase++**

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oGauge
```

```
    LOCAL oLayer,oLayer1
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oGauge := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-83ED1790AAD3}*/
```

```
    oGauge:create(,, {10,60},{610,370} )
```

```
        oGauge:BeginUpdate()
```

```
        oGauge:SetProperty("DefaultLayer",185/*exDefLayerRotateType*/,2)
```

```
        oGauge:SetProperty("BackColor",AutomationTranslateColor( GraMakeRGBColor  
( { 217,217,217 } ) , .F. ))
```

```
        oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
        oLayer := oGauge:Layers():Add("background")
```

```
            oLayer:Background():Picture():Name := "Guage_Background.png"
```

```
            oLayer:RotateCenterY := "lheight/2 + 78"
```

```
        oLayer1 := oGauge:Layers():Add("needle")
```

```
            oLayer1:Background():Picture():Name := "Guage_Needle.png"
```

```
            oLayer1:OnDrag := 2/*exDoRotate*/
```

```
            oLayer1:RotateAngleValid := "value < 90 ? value : (value < 180 ? 90 : ( value <  
270 ? 270 : value ))"
```

```
            oLayer1:RotateAngleToValue := "value >= 270 ? (value - 270)/90*50 :  
(value/90)*50 + 50"
```

```
            oLayer1:ValueToRotateAngle := "value < 50 ? (270 + value/50*90) : (value -  
50)/50 * 90"
```

```
        oGauge:Value := 85
```

```
oGauge:EndUpdate()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

# property Gauge.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 Gauge.VisualAppearance as Appearance

Retrieves the control's appearance.

Type	Description
<a href="#">Appearance</a>	An Appearance object that holds a collection of skins.

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

# Layer object

The Layer object holds a collection of pictures and HTML captions to be displayed on a viewable layer. Any layer can be visible, selectable or draggable. Any layer can be moved, rotated or clipped. The [Item](#) property of the Layers collection access a Layer object of the control. The [Background](#) / [Foreground](#) defines the layer's background and foreground.

The following screen shot shows combination of pictures and HTML captions, on different controls:



The Layer object supports the following properties and methods:

Name	Description
<a href="#">Background</a>	Gets access to the layer's background object.
<a href="#">Brightness</a>	Specifies the percent of brightness to apply to the layer.
<a href="#">Clip</a>	Gets access to the layer's clip object.
<a href="#">Contrast</a>	Specifies the percent of contrast to apply to the layer.
<a href="#">DefaultOffsetX</a>	Gets or sets a value that indicates the default x-offset of the layer.
<a href="#">DefaultOffsetY</a>	Gets or sets a value that indicates the default y-offset of the layer.
<a href="#">DefaultRotateAngle</a>	Specifies the default angle to rotate the layer.
<a href="#">Foreground</a>	Gets access to the layer's foreground object.
<a href="#">Grayscale</a>	Returns or sets a value that indicates that indicates percent to convert the layer to grayscale.
<a href="#">Height</a>	Specifies the expression relative to the view, to determine the height to show the current layer on the control.

<a href="#">Idem</a>	Ensures that the layer's offset and rotation-angle is equal for all idem layers (separated by comma character).
<a href="#">Index</a>	Indicates the index of the layer.
<a href="#">Key</a>	Retrieves or sets the layer's key.
<a href="#">LayerClipToAlpha</a>	Returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region.
<a href="#">LayerToClientX</a>	Converts the x-position of the layer to control's client x-position.
<a href="#">LayerToClientY</a>	Converts the x-position of the layer to control's client x-position.
<a href="#">Left</a>	Specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
<a href="#">OffsetToValue</a>	Specifies the expression to convert the offsetX,offsetY to value.
<a href="#">OffsetX</a>	Gets or sets a value that indicates x-offset of the layer.
<a href="#">OffsetXValid</a>	Validates the x-offset value of the layer
<a href="#">OffsetY</a>	Gets or sets a value that indicates y-offset of the layer.
<a href="#">OffsetYValid</a>	Validates the y-offset value of the layer
<a href="#">OnDrag</a>	Indicates the action to be performed when the user drags the layer.
<a href="#">Position</a>	Retrieves or sets a value that indicates the position/z-order of the layer in the control.
<a href="#">RotamoveCenterX</a>	Specifies the x-position of the layer's center, while the layer's drag operation is exDoRotamove.
<a href="#">RotamoveCenterY</a>	Specifies the y-position of the layer's center, while the layer's drag operation is exDoRotamove.
<a href="#">RotamoveOffsetX</a>	Specifies the x-offset of the layer, while the layer's drag operation is exDoRotamove.
<a href="#">RotamoveOffsetY</a>	Specifies the y-offset of the layer, while the layer's drag operation is exDoRotamove.
<a href="#">RotateAngle</a>	Specifies the angle to rotate the layer.
<a href="#">RotateAngleToValue</a>	Specifies the expression to convert the rotating angle to value.
<a href="#">RotateAngleValid</a>	Validates the rotation angle of the layer.
	Indicates the index of the layer the rotation is around. If -1,

<a href="#">RotateCenterLayer</a>	the rotation is relative to the current layer.
<a href="#">RotateCenterX</a>	Indicates the expression that determines the x-origin of the rotation point relative to the RotateCenterLayer layer.
<a href="#">RotateCenterY</a>	Indicates the expression that determines the y-origin of the rotation point relative to the RotateCenterLayer layer.
<a href="#">RotateClip</a>	Specifies whether the layer's clipping region is rotated once the layer is rotated.
<a href="#">RotateType</a>	Returns or sets a value that indicates whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ...
<a href="#">Selectable</a>	Returns or sets a value that indicates whether the layer is selectable.
<a href="#">ShowHandCursor</a>	Returns or sets a value that indicates whether the hand cursor is shown when it hovers a visible / selectable / draggable layer.
<a href="#">ToolTip</a>	Gets or sets a value (tooltip) that's displayed once the cursor hovers the layer.
<a href="#">ToolTipTitle</a>	Gets or sets a value (title) that's displayed once the cursor hovers the layer.
<a href="#">Top</a>	Specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
<a href="#">Transparency</a>	Gets or sets a value that indicates percent of the transparency to display the layer.
<a href="#">UserData</a>	Indicates any extra data associated with the layer.
<a href="#">Value</a>	Indicates the object's value.
<a href="#">ValueToOffsetX</a>	Specifies the expression to convert the value to x-offset.
<a href="#">ValueToOffsetY</a>	Specifies the expression to convert the value to y-offset.
<a href="#">ValueToRotateAngle</a>	Specifies the expression to convert the value to rotating angle
<a href="#">Visible</a>	Retrieves or sets a value indicating whether the layer is visible or hidden.
<a href="#">Width</a>	Specifies the expression relative to the view, to determine the width to show the current layer on the control.

# property Layer.Background as Background

Gets access to the layer's background object.

Type	Description
Background	A <a href="#">Background</a> object that indicates the layer's background.

The Background property gets access to the layer's background object. The [Background](#) object holds pictures to be shown on the layer's background. The Layer's background can display unlimited graphics of different sizes and positions. For instance, the [Picture](#) property of the Background specifies the picture to be shown on the layer's background. Use the [ExtraPicture](#) property to assign a new picture on the same location. The [Color](#) property indicates the layer's color object, so you can apply a solid color on the layer's background.

The following screen shot shows pictures on each layer's background:



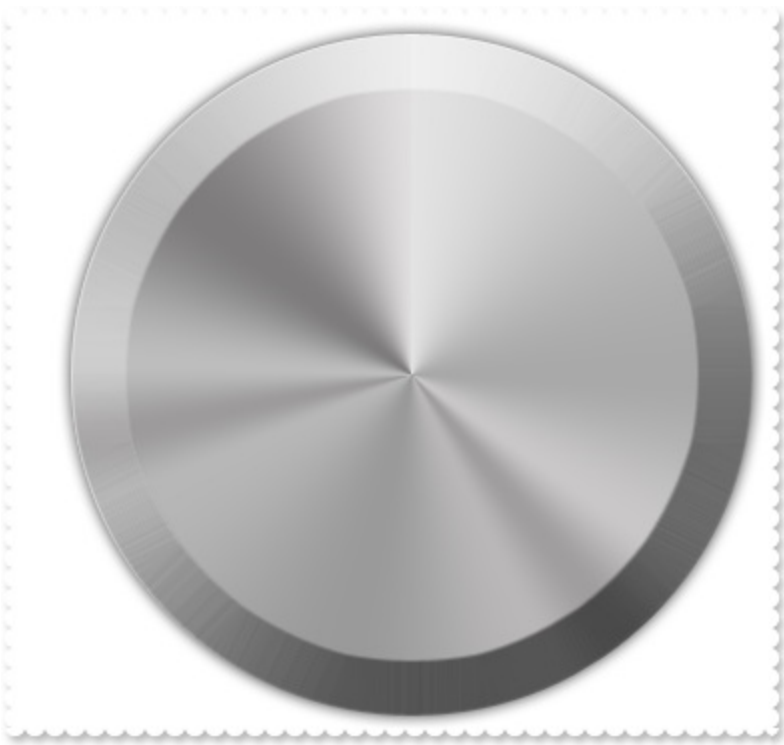
# property Layer.Brightness(Channel as ColorAdjustmentChannelEnum) as Long

Specifies the percent of brightness to apply to the layer.

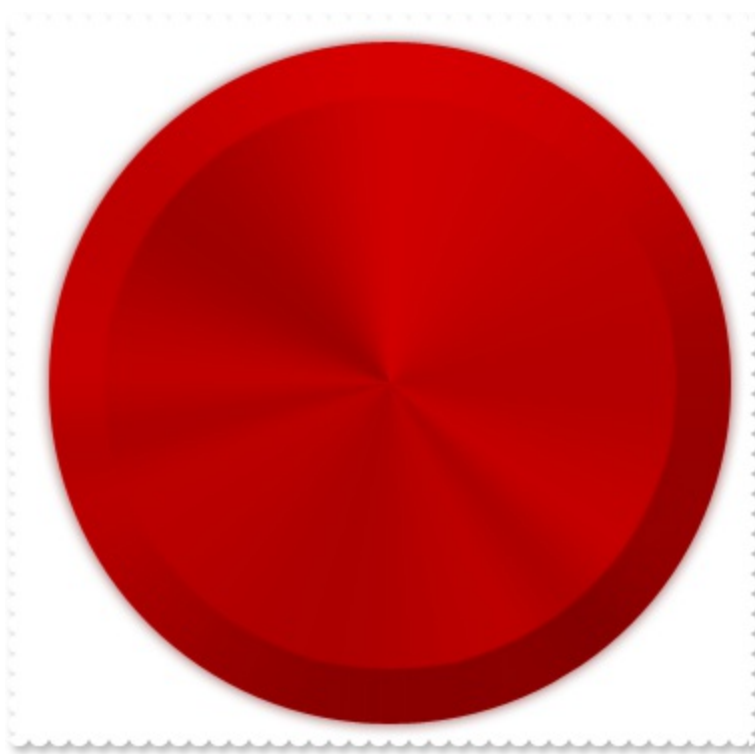
Type	Description
Channel as <a href="#">ColorAdjustmentChannelEnum</a>	A ColorAdjustmentChannelEnum expression that specifies the channel to be changed.
Long	A Long expression that specifies the percent of brightness / color to apply to the layer.

By default, the Brightness on all channels is 50%, which indicates that no effect is applied to the layer. The Brightness specifies the percent of brightness to apply to the layer. Use the [DefaultLayer\(exDefLayerBrightness...\)](#) property to specify the default value for the Brightness(exAllChannels...), before adding any layer. The Brightness / [Contrast](#) properties can be used to change the percent of specified color to be applied on the layer. You can use the [Grayscale](#) property to show the entire layer in gray scale ( disable state).

The following screen shot shows the layer, with all Brightness properties on 50% ( default ):



The following screen shot shows the layer, with more red, Brightness(exAllChannels) = 0,Brightness(exRedChannel) = 75:



By default, the [AllowSmoothChange](#) property is `exLayerTransparency | exLayerBrightness | exLayerContrast`. Use the `AllowSmoothChange` property to disable changing gradually any brightness / contrast or the transparency, of the layer. For instance, a gradually change means that you can change the layer's transparency from 0 to 50 in a short time, with intermediate values ( smooth change ).

The `AllowSmoothChange` property changes gradually one / or more properties like follow:

- `Brightness`, Specifies the percent of brightness to apply to the layer.
- [Contrast](#), Specifies the percent of contrast to apply to the layer.
- [Transparency](#), Gets or sets a value that indicates percent of the transparency to display the layer.

The [MouseIn](#) / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects. The `AllowSmoothChange` property specifies the properties of the layers that support smooth change. For instance, you can use the `MouseIn` / `MouseOut` event to change gradually the brightness / contrast or the transparency, of the layer, while the `AllowSmoothChange` property is not `exSmoothChangeless`.

# property Layer.Clip as Clip

Gets access to the layer's clip object.

Type	Description
Clip	A <a href="#">Clip</a> object that helps you to clip the current layer.

The Clip property accesses the layer's Clip object. The Clip object defines the clipping you can apply to any layer on the control. The Clipping support include intersection of any of rectangle, round rectangle, ellipse, pie, picture mask, polygon, and so on. The [RotateClip](#) property specifies whether the layer's clipping region is rotated once the layer is rotated.

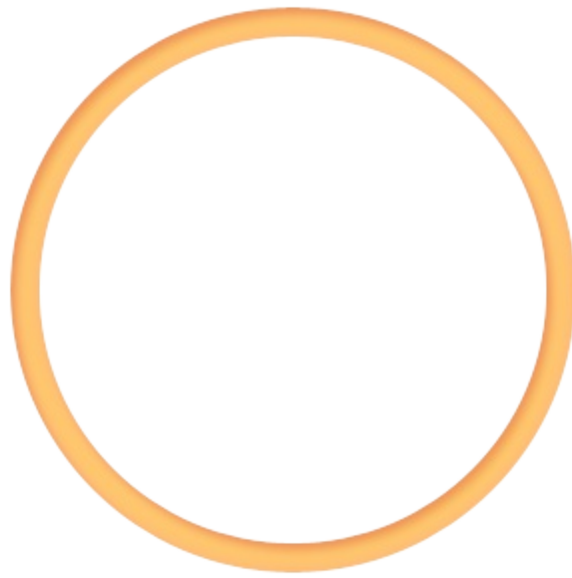
Any of the following properties ( or combination of them ) can be used to do the clipping:

- [Ellipse](#), clips the layer as a ellipse / circle
- [Picture](#), clips the layer using a picture as a mask
- [Pie](#), clips the layer as a arc / pie
- [Polygon](#), clips the layer giving the points that define a polygon, triangle, rectangle, and so on
- [Rectangle](#), clips the layer giving a rectangle
- [RoundRectangle](#), clips the layer giving a round rectangle

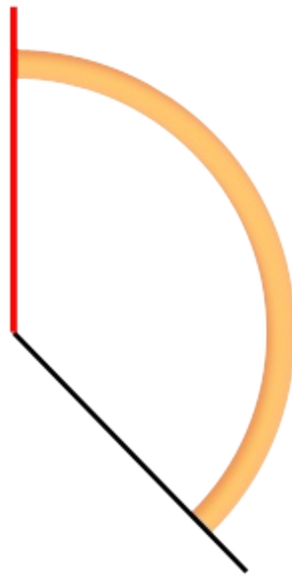
The [Type](#) property specifies the type of the clipping the current layer supports. For instance `LayerClipTypeEnum.exLayerClipPie | LayerClipTypeEnum.exLayerClipRectangle` specifies that a pie and rectangle clippings are combined together. The [Value](#) property of the Clip indicates the value to be applied to the current clipping object to define a new shape based on the value. For instance, you may want to define a value from 0 to 100, for a circle, and for 50 to show a half of circle, for 25, a quarter of circle ( pie ), and so on.

Having the following layer:





By clipping, we can get something like follows:



and if we display the entire gauge here's what we get:



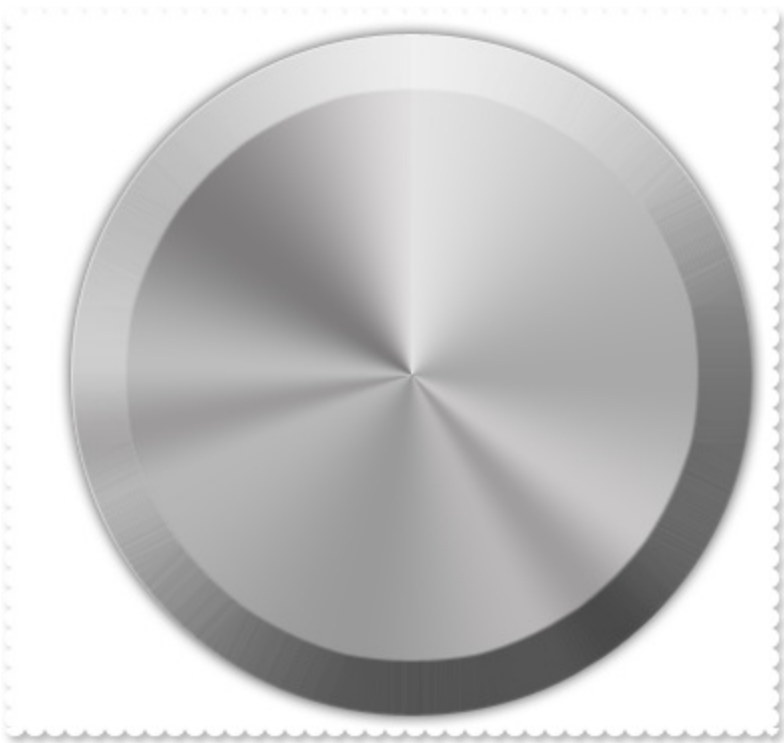
# property Layer.Contrast(Channel as ColorAdjustmentChannelEnum) as Long

Specifies the percent of contrast to apply to the layer.

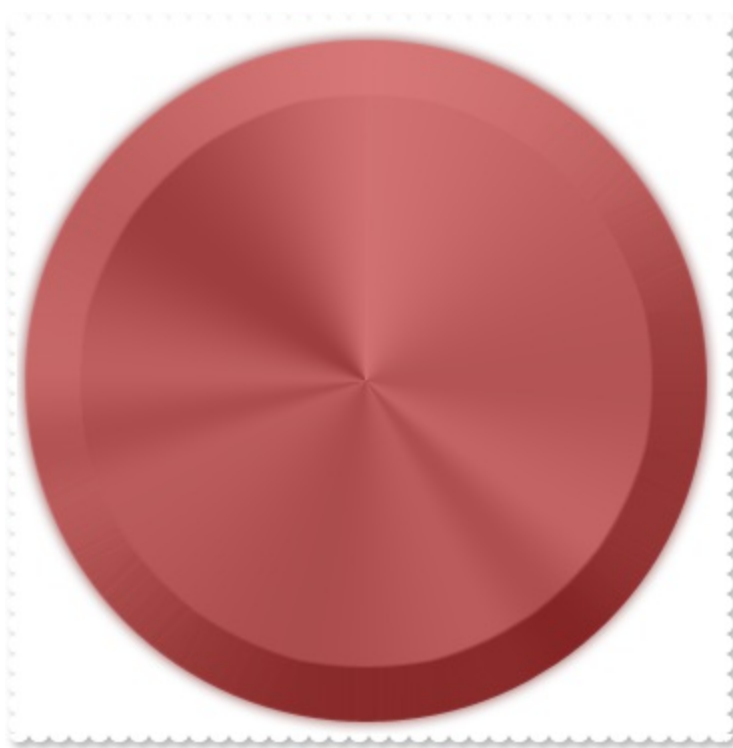
Type	Description
Channel as <a href="#">ColorAdjustmentChannelEnum</a>	A ColorAdjustmentChannelEnum expression that specifies the channel to be changed.
Long	A Long expression that specifies the percent of contrast / color to apply to the layer.

By default, the Contrast on all channels is 50%, which indicates that no effect is applied to the layer. The Contrast property specifies the percent of contrast to apply to the layer. Use the [DefaultLayer\(exDefLayerContrast...\)](#) property to specify the default value for the Contrast(exAllChannels...), before adding any layer. The [Brightness](#) / Contrast properties can be used to change the percent of specified color to be applied on the layer. You can use the [Grayscale](#) property to show the entire layer in gray scale ( disable state).

The following screen shot shows the layer, with all Contrast properties on 50% ( default ):



The following screen shot shows the layer, with more red, Contrast(exAllChannels) = 0, Contrast(exRedChannel) = 75:



By default, the [AllowSmoothChange](#) property is `exLayerTransparency | exLayerBrightness | exLayerContrast`. Use the `AllowSmoothChange` property to disable changing gradually any brightness / contrast or the transparency, of the layer. For instance, a gradually change means that you can change the layer's transparency from 0 to 50 in a short time, with intermediate values ( smooth change ).

The `AllowSmoothChange` property changes gradually one / or more properties like follow:

- [Brightness](#), Specifies the percent of brightness to apply to the layer.
- `Contrast`, Specifies the percent of contrast to apply to the layer.
- [Transparency](#), Gets or sets a value that indicates percent of the transparency to display the layer.

The [MouseIn](#) / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects. The `AllowSmoothChange` property specifies the properties of the layers that support smooth change. For instance, you can use the `MouseIn` / `MouseOut` event to change gradually the brightness / contrast or the transparency, of the layer, while the `AllowSmoothChange` property is not `exSmoothChangeless`.

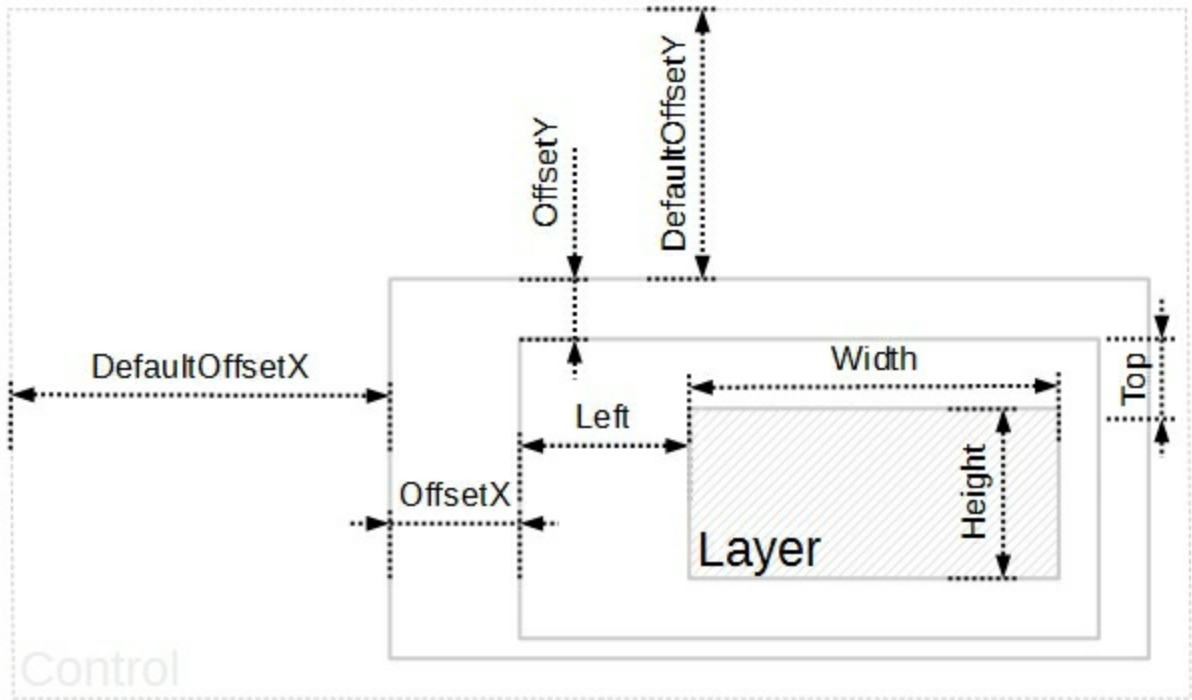
# property Layer.DefaultOffsetX as Long

Gets or sets a value that indicates the default x-offset of the layer.

Type	Description
Long	A Long expression that defines the default x-offset of the layer.

By default, the DefaultOffsetX property is 0. The DefaultOffsetX property gets or sets a value that indicates the default x-offset of the layer. You can use the DefaultOffsetX / DefaultOffsetX properties to move the layer to a different position, which could be the initial position. Use the [DefaultLayer\(exDefLayerDefaultOffsetX\)](#) property to specify the default value for the DefaultOffsetX, before adding any layer.

The following picture shows the position/size properties of the Layer, relative to the view / control:



The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- `DefaultOffsetX`, gets or sets a value that indicates the default x-offset of the layer.
- [`OffsetX`](#), gets or sets a value that indicates x-offset of the layer.
- [`OffsetXValid`](#), validates the x-offset value of the layer.
- [`Value`](#) and [`ValueToOffsetX`](#) specifies the expression to convert the value to x-offset.
- [`DefaultOffsetY`](#), gets or sets a value that indicates the default y-offset of the layer.
- [`OffsetY`](#), gets or sets a value that indicates y-offset of the layer.
- [`OffsetYValid`](#), validates the y-offset value of the layer.
- [`Value`](#) and [`ValueToOffsetY`](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [`DisplayAs`](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [`Left`](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [`Top`](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [`Width`](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [`Height`](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

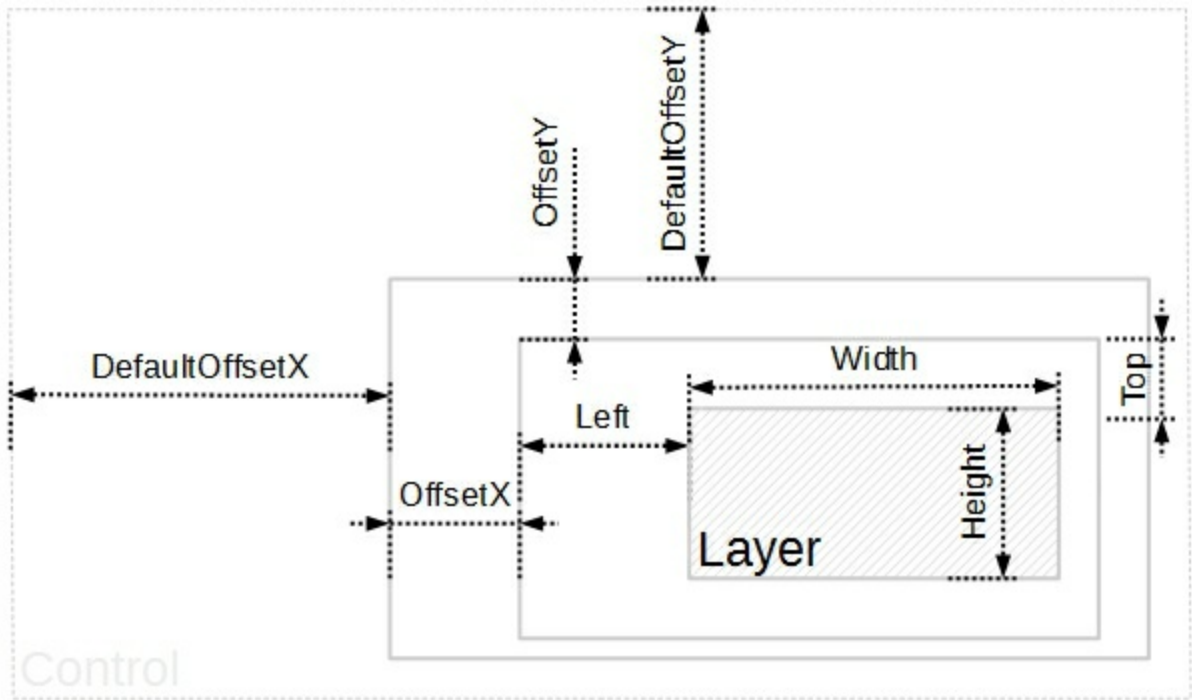
# property Layer.DefaultOffsetY as Long

Gets or sets a value that indicates the default y-offset of the layer.

Type	Description
Long	A Long expression that defines the default y-offset of the layer.

By default, the DefaultOffsetY property is 0. The DefaultOffsetY property gets or sets a value that indicates the default y-offset of the layer. You can use the DefaultOffsetX / DefaultOffsetX properties to move the layer to a different position, which could be the initial position. Use the [DefaultLayer\(exDefLayerDefaultOffsetY\)](#) property to specify the default value for the DefaultOffsetY, before adding any layer.

The following picture shows the position/size properties of the Layer, relative to the view / control:



The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.



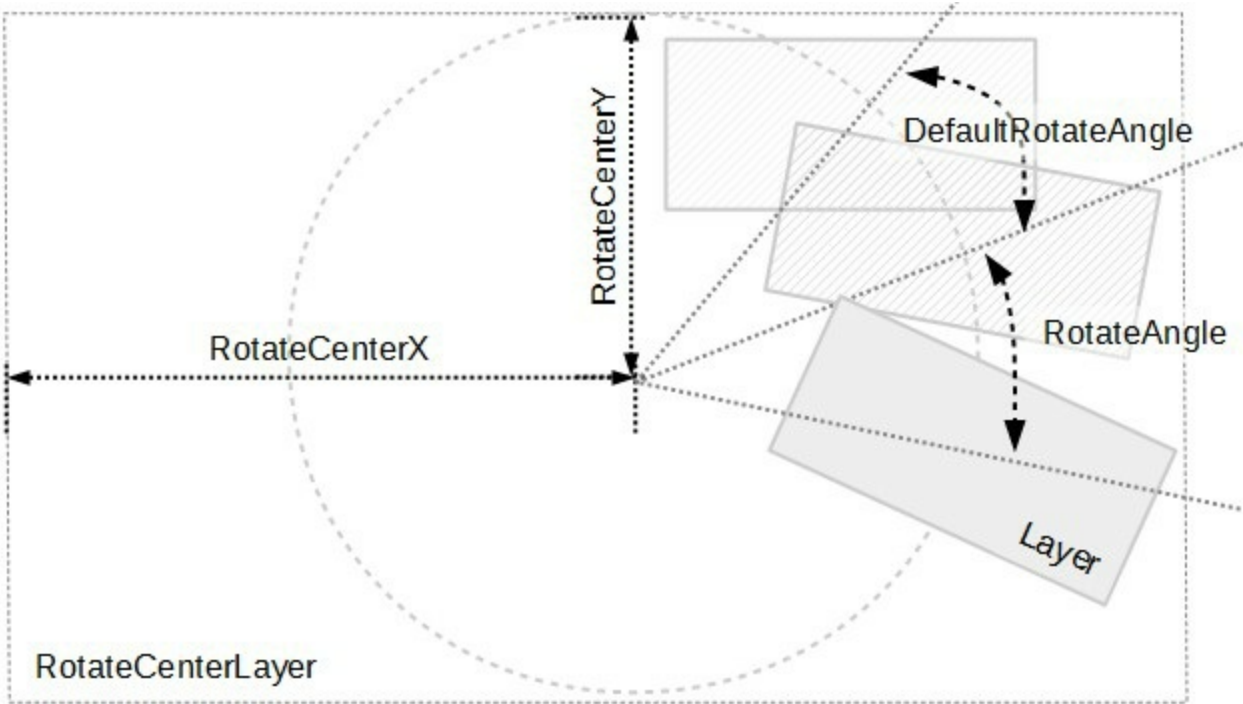
# property Layer.DefaultRotateAngle as Double

Specifies the default angle to rotate the layer.

Type	Description
Double	A Double expression that specifies the default angle to rotate the layer, in degree.

By default, the DefaultRotateAngle property is 0 degree ( which indicates that the layer is shown as it is ). The DefaultRotateAngle property specifies the default angle to rotate the layer. For instance, you can use the DefaultRotateAngle property to show the current layer in a different position, as an initial view. Use the [DefaultLayer\(exDefLayerDefaultRotateAngle\)](#) property to specify the default value for the DefaultRotateAngle property, before adding any layer. The [RotateType](#) property specifies whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ... Change the [Debug](#) property of the [DragInfo](#) during the [DragStart](#) event to debug the rotation angles.

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- DefaultRotateAngle, specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

# property Layer.Foreground as Foreground

Gets access to the layer's foreground object.

Type	Description
Foreground	A Foreground object that holds the HTML captions to be shown on the layer's foreground.

The Foreground object holds HTML captions to be shown on the layer's foreground. The [Background](#) object holds pictures to be shown on the layer's background. The Layer's foreground can display unlimited HTML captions of different sizes and positions. The [Caption](#) property specifies the caption on the layer. The [ExtraCaption](#) property assigns a new HTML caption on the layer's foreground. The [Color](#) property specifies the layer's foreground color.

The following screen shot shows all layer's background with a semi-transparent color, to highlight the layer's foreground:



# property Layer.Grayscale as Long

Returns or sets a value that indicates whether the layer is show as grayscale.

Type	Description
Long	A long expression that indicates the percent to convert the layer into grayscale (value between 0 and 100)

By default, the Grayscale property is 0, so the layer is shown normal ( enabled ). You can use the Grayscale property to show the entire layer in gray scale ( disable state). Use the [DefaultLayer\(exDefLayerGrayscale\)](#) property to specify the default value for the Grayscale property, before adding any layer. The [Brightness](#) / [Contrast](#) properties can be used to change the percent of specified color to be applied on the layer. The [Selectable](#) property specifies whether the user can select the layer at runtime. For instance, you can simulate a disabled layer by changing the layer's Grayscale property on True, and setting the layer's Selectable property on False.

The following screen shot shows the layer, with Grayscale property on False ( by default ):



The following screen shot shows the layer, with Grayscale property on True:



## property Layer.Height as String

Specifies the expression relative to the view, to determine the height to show the current layer on the control.

Type	Description
String	A String value that specifies the expression relative to the view, to determine the height to show the current layer on the control.

By default, the Height property is "height". If the Height property is empty, missing or invalid, it is considered "height". If valid, the value of evaluating the Height property indicates the height of the layer as shown in the picture bellow. Use the [DefaultLayer\(exDefLayerHeight\)](#) property to specify the default value for the Height property, before adding any layer. The [LayerAutoSize](#) property resizes all layers based on the picture of the first layer.

For instance:

- "0" indicates that the layer's height is 0
- "height / 2", half of the view or center of the control's view
- "height - 64", 64 pixels to the bottom side of the control's view

The Height property supports the following keywords:

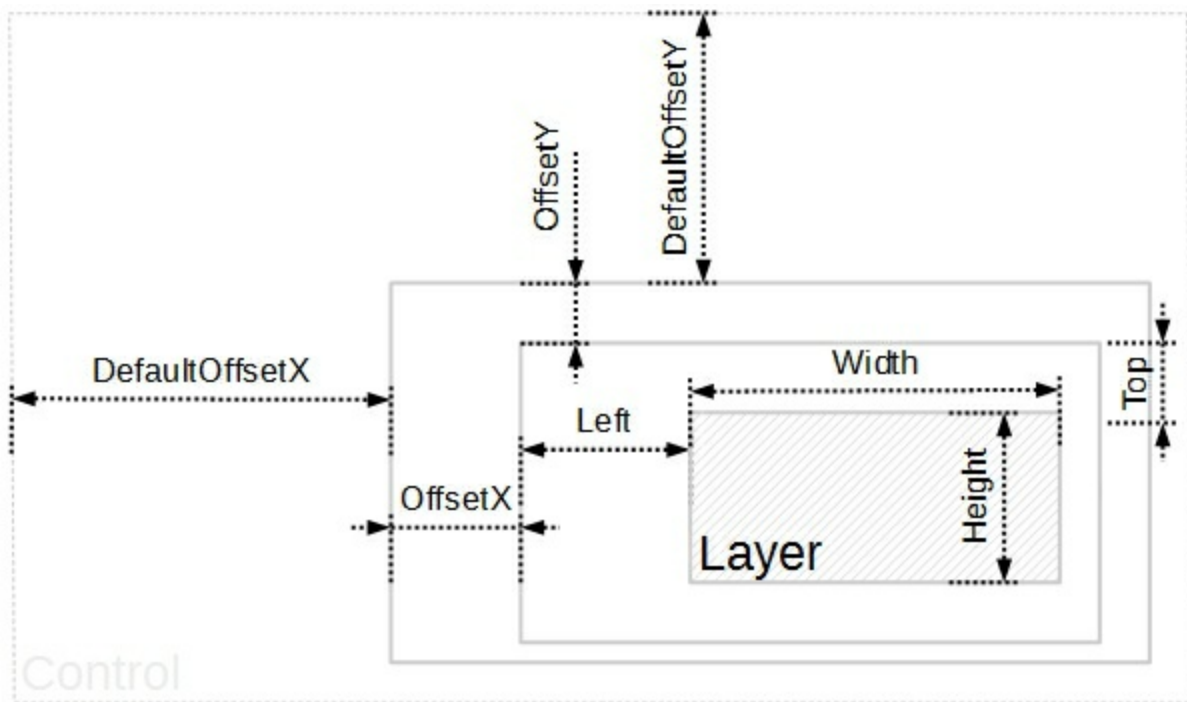
- **width** keyword specifies the width in pixels of the control's view
- **height** keyword specifies the height in pixels of the control's view

Also, this property supports all constants, operators and functions defined [here](#).

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- Height, specifies the expression relative to the view, to determine the height to show the current layer on the control.

The following picture shows the position/size properties of the Layer, relative to the view / control:



You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

## property Layer.Idem as Variant

Ensures that the layer's offset and rotation-angle is equal for all idem layers (separated by comma character).

Type	Description
Variant	A long expression that specifies the index of the layer idem with current layer, a string expression that specifies the key of the layer idem with the current layer, or a list of keys separated by comma character. For instance, Idem = "0,1" indicates that the current layer's rotation-angle and offset are always the same as layers with the index 0 and 1

By default, the Idem property is empty, which indicates hat has no effect. You can use the Idem property to rotate or move multiple layers once user drags a layer. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Key](#) property retrieves or sets the layer's key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index [Count](#) - 1, is the last layer in the control ( in z-order ).



# property Layer.Index as Long

Indicates the index of the layer.

Type	Description
Long	A Long expression that specifies the index of the layer.

By default, the Index property is automatically updated by the control, as soon as the user adds / removes layers to the control. The Index property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index [Count](#) - 1, is the last layer in the control ( in z-order ). The [Key](#) property retrieves or sets the layer's key. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [LayerFromPoint](#) retrieves the index of the layer from the point ( only visible and selectable objects are included ).

# property Layer.Key as Variant

Retrieves or sets the layer's key.

Type	Description
Variant	A VARIANT expression that specifies the key to identify the layer.

By default, the Key property is empty. The Key property retrieves or sets the layer's key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index [Count](#) - 1, is the last layer in the control ( in z-order ). The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [LayerFromPoint](#) retrieves the index of the layer from the point ( only visible and selectable objects are included ).

## property Layer.LayerClipToAlpha as Long

Returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region.

Type	Description
Long	A Long expression that returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region.

By default, the LayerClipToAlpha property is 0, which indicates that only pixels of the layer that has 0 on the alpha channel (transparent pixels) defines the transparent region, and so the clipping region. The LayerClipToAlpha property property of the Layer object, returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region. In other words, the value from 0 to LayerClipToAlpha defines transparent pixels, and the rest defines the opaque pixels to be included in the clipping region. So based on the layer's picture, you can change the LayerClipToAlpha property for a better look of your widget. The [LayerClipTo](#) property specifies the index of the layer that clips the entire control to. The [LayerClipToParent](#) property indicates if the [LayerClipTo](#) method clips the control itself, parent or the owner of the control.

"I would like to put the control on a form, then make the form transparent so the control appears on the desktop with just the images contained in the layers visible. For example, take a clock example and make the control background and the form transparent, and you have a working clock widget."

The control support transparent form, or in other words, displaying the control's itself without its form behind. In order to make your eXGauge control to display a widget, ( no form behind or form transparent ), you need to use the following properties:

- [LayerClipTo](#) property of the control, specifies the index of the layer that clips the entire control to. By default, the LayerClipTo property is -1, which indicates that no clipping is supported. So, one of the layers that composes your widget must be specified as the widget's background, and so, the entire view of the control is clipped to region defined by the clipping layer (LayerClipTo). The LayerClipTo property may refer to any layer, visible or hidden, which includes a picture or a clipping object ( Clip property ).
- **Layer.LayerClipToAlpha** property of the Layer object, returns or sets a value that indicates the value of the alpha channel to be included in the LayerClipTo region. By default, the LayerClipToAlpha property is 0, which indicates that only pixels of the layer that has 0 on the alpha channel (transparent pixels) defines the transparent region, and so the clipping region. In other words, the value from 0 to LayerClipToAlpha defines transparent pixels, and the rest defines the opaque pixels to be included in the clipping region. Sobased on the layer's picture, you can change the LayerClipToAlpha property

for a better look of your widget.

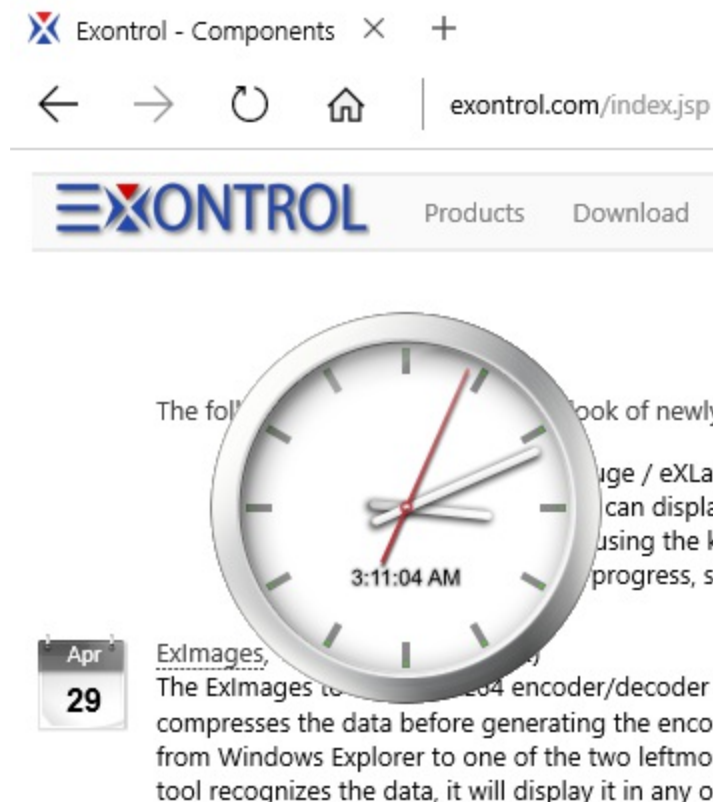
- **LayerClipToParent** property of the control, indicates if the LayerClipTo method clips the control itself, parent or the owner of the control. By default, the LayerClipToParent property is exLayerUpdateControl, which indicates that the control's itself is clipped relative to its form that hosts it. Change the LayerClipToParent property to exLayerUpdateScreen, or exLayerUpdateParent, and so the clipping region is applied to its form/dialog/parent window.

The following VB sample defines the control as a widget:

```
With Gauge1
    .LayerClipTo = 0
    .LayerClipToParent = exLayerUpdateScreen
End With
```

The sample defines the layer with the Index 0, as being the clipping layer. The setup installs the C:\Program Files\Exontrol\ExGauge\Sample\VB\Clock-Widget-Region that shows all these working.

The following screen shot shows the control on a transparent form:



The following screen shot shows the control on an opaque form:

**EXONTROL**

Products

Download

http://www.e... — □ ×



newl

eXL2

displ

the

ess, !

oder

encc

eftmc

tool recognizes the data, it will display it in any c

## property Layer.LayerToClientX (X as Variant, Y as Variant) as Long

Converts the x-position of the layer to control's client x-position.

Type	Description
X as Variant	A Lone expression that specifies the x-position of the point within the layer
Y as Variant	A Lone expression that specifies the y-position of the point within the layer
Long	A Long expression that specifies the x-position of the point on the control's view that's equivalent of the point on the layer.

The LayerToClientX / [LayerToClientY](#) converts the (x,y)-point on the layer to control's view point. The LayerToClientX / [LayerToClientY](#) properties translate a point from the layer ( as it is moved or rotated ) to the control's view. For instance, you can display the current value of the control on the knob you are rotating. The [RotamoveCenterX](#) / [RotamoveCenterY](#) specifies the (x,y)-position of the layer's center, while the layer's drag operation is exDoRotamove. The [OnDrag](#) property indicates the action to be performed when the user drags the layer.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.



The following sample shows how you can use the LayerToClientX / [LayerToClientY](#) properties to display the layer's value on the knob:

### VBA (MS Access, Excell...)

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)
```

```
    With Gauge1
```

```
        .ExtraCaption("Client",0) = .FormatABC("`<sha ;;0><font ;12><b>` + (100 - value  
format `0`)",Gauge1.Value)
```

```
        .ExtraCaption("Client",4) = .FormatABC("value -  
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gau
```

```
        .ExtraCaption("Client",5) = .FormatABC("value -  
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau
```

```
    End With
```

```
End Sub
```

```
With Gauge1
```

```
    .BeginUpdate
```

```
    .PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```

.PicturesName = "`Layer` + str(value + 1)+ `.png`"
.Layers.Count = 11
.AllowSmoothChange = 0
With .Layers.Item(9)
    .DefaultRotateAngle = -126
    .OnDrag = 3
    .RotateAngleToValue = "100 - (value / 360 * 100)"
    .ValueToRotateAngle = "(value)/100 * 360"
    .ValueToOffsetX = "value"
    .OffsetToValue = "value"
    .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
End With
With .Layers.Item(7)
    .OnDrag = 2
    .RotateType = 2
End With
.Value = 25
.EndUpdate
End With

```

## VB6

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)
```

```
    With Gauge1
```

```
        .ExtraCaption("Client",exLayerCaption) = .FormatABC("`<sha ;;0> <font ;12> <b>`  
+ (100 - value format `0`)",Gauge1.Value)
```

```
        .ExtraCaption("Client",exLayerCaptionLeft) = .FormatABC("value -  
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gaug
```

```
        .ExtraCaption("Client",exLayerCaptionTop) = .FormatABC("value -  
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau
```

```
    End With
```

```
End Sub
```

```
With Gauge1
```



```

.BeginUpdate
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + str(value + 1) + `.png`"
.Layers.Count = 11
.AllowSmoothChange = exSmoothChangeless
With .Layers.Item(9)
.DefaultRotateAngle = -126
.OnDrag = exDoRotamove
.RotateAngleToValue = "100 - (value / 360 * 100)"
.ValueToRotateAngle = "(value)/100 * 360"
.ValueToOffsetX = "value"
.OffsetToValue = "value"
.RotateAngleValid = "int(value / 360 * 100)/100 * 360"
End With
With .Layers.Item(7)
.OnDrag = exDoRotate
.RotateType = exRotateBilinearInterpolation
End With
.Value = 25
.EndUpdate
End With

```

## VB.NET

*' Change event - Occurs when the layer's value is changed.*

```

Private Sub Exgauge1_Change(ByVal sender As System.Object,ByVal Layer As Integer)
Handles Exgauge1.Change
    With Exgauge1

.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
;;0> <font ;12> <b>` + (100 - value format `0`)",Exgauge1.Value))

.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
-
8",Exgauge1.Layers.Item(9).get_LayerToClientX(Exgauge1.Layers.Item(9).RotamoveCent

```

```

.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
-
26",Exgauge1.Layers.Item(9).get_LayerToClientY(Exgauge1.Layers.Item(9).RotamoveCen

    End With
End Sub

With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + str(value + 1) + `.png`"
    .Layers.Count = 11
    .AllowSmoothChange =
exontrol.EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless
    With .Layers.Item(9)
        .DefaultRotateAngle = -126
        .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotamove
        .RotateAngleToValue = "100 - (value / 360 * 100)"
        .ValueToRotateAngle = "(value)/100 * 360"
        .ValueToOffsetX = "value"
        .OffsetToValue = "value"
        .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
    End With
    With .Layers.Item(7)
        .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
        .RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
    End With
    .Value = 25
    .EndUpdate()
End With

```

## VB.NET for /COM

' Change event - Occurs when the layer's value is changed.

Private Sub AxGauge1\_Change(ByVal sender As System.Object, ByVal e As AxEXGAUGELib.\_IGaugeEvents\_ChangeEvent) Handles AxGauge1.Change  
With AxGauge1

.set\_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,.Fo  
;;0> <font ;12> <b>` + (100 - value format `0`)",AxGauge1.Value))

.set\_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft  
-  
8",AxGauge1.Layers.Item(9).LayerToClientX(AxGauge1.Layers.Item(9).RotamoveCenterX,

.set\_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop  
-  
26",AxGauge1.Layers.Item(9).LayerToClientY(AxGauge1.Layers.Item(9).RotamoveCenter,

End With  
End Sub

With AxGauge1

.BeginUpdate()  
.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
.PicturesName = "`Layer` + str(value + 1) + `.png`"  
.Layers.Count = 11  
.AllowSmoothChange = EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless  
With .Layers.Item(9)  
.DefaultRotateAngle = -126  
.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotamove  
.RotateAngleToValue = "100 - (value / 360 \* 100)"  
.ValueToRotateAngle = "(value)/100 \* 360"  
.ValueToOffsetX = "value"  
.OffsetToValue = "value"  
.RotateAngleValid = "int(value / 360 \* 100)/100 \* 360"

End With

With .Layers.Item(7)

.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate

```

.RotateType = EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
End With
.Value = 25
.EndUpdate()
End With

```

## C++

```

// Change event - Occurs when the layer's value is changed.
void OnChangeGauge1(long Layer)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
        Library'
        #import <ExGauge.dll>
        using namespace EXGAUGELib;
    */
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    spGauge1->PutExtraCaption("Client",EXGAUGELib::exLayerCaption,spGauge1-
> FormatABC(L"<sha ;;0> <font ;12> <b>` + (100 - value format `0`)",spGauge1-
> GetValue(),vtMissing,vtMissing));
    spGauge1->PutExtraCaption("Client",EXGAUGELib::exLayerCaptionLeft,spGauge1-
> FormatABC(L"value - 8",spGauge1->GetLayers()->GetItem(long(9))-
> GetLayerToClientX(spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterX(),spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterY()),vtMissing,vtMissing));
    spGauge1->PutExtraCaption("Client",EXGAUGELib::exLayerCaptionTop,spGauge1-
> FormatABC(L"value - 26",spGauge1->GetLayers()->GetItem(long(9))-
> GetLayerToClientY(spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterX(),spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterY()),vtMissing,vtMissing));
}

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();

```

```

spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + str(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(11);
spGauge1->PutAllowSmoothChange(EXGAUGELib::exSmoothChangeless);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(long(9));
    var_Layer->PutDefaultRotateAngle(-126);
    var_Layer->PutOnDrag(EXGAUGELib::exDoRotamove);
    var_Layer->PutRotateAngleToValue(L"100 - (value / 360 * 100)");
    var_Layer->PutValueToRotateAngle(L"(value)/100 * 360");
    var_Layer->PutValueToOffsetX(L"value");
    var_Layer->PutOffsetToValue(L"value");
    var_Layer->PutRotateAngleValid(L"int(value / 360 * 100)/100 * 360");
EXGAUGELib::ILayerPtr var_Layer1 = spGauge1->GetLayers()->GetItem(long(7));
    var_Layer1->PutOnDrag(EXGAUGELib::exDoRotate);
    var_Layer1->PutRotateType(EXGAUGELib::exRotateBilinearInterpolation);
spGauge1->PutValue(long(25));
spGauge1->EndUpdate();

```

## C++ Builder

```

// Change event - Occurs when the layer's value is changed.
void __fastcall TForm1::Gauge1Change(TObject *Sender,long Layer)
{
    Gauge1-
> ExtraCaption[TVariant("Client"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCa
    = TVariant(Gauge1->FormatABC(L"<sha ;;0> <font ;12> <b>` + (100 - value format
`0`)",TVariant(Gauge1->get_Value()),TNoParam(),TNoParam()));
    Gauge1-
> ExtraCaption[TVariant("Client"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCa
    = TVariant(Gauge1->FormatABC(L"value - 8",TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->get_LayerToClientX(TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterX),TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterY))),TNoParam(),TNoParam()));
    Gauge1-

```

```

> ExtraCaption[TVariant("Client"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCa
= TVariant(Gauge1->FormatABC(L"value - 26",TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->get_LayerToClientY(TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterX),TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterY))),TNoParam(),TNoParam()));
}

Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + str(value + 1) + `.png`;
Gauge1->Layers->Count = 11;
Gauge1->AllowSmoothChange =
Exgaugelib_tlb::SmoothPropertyEnum::exSmoothChangeless;
Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(9));
var_Layer->DefaultRotateAngle = -126;
var_Layer->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotamove;
var_Layer->RotateAngleToValue = L"100 - (value / 360 * 100)";
var_Layer->ValueToRotateAngle = L"(value)/100 * 360";
var_Layer->ValueToOffsetX = L"value";
var_Layer->OffsetToValue = L"value";
var_Layer->RotateAngleValid = L"int(value / 360 * 100)/100 * 360";
Exgaugelib_tlb::ILayerPtr var_Layer1 = Gauge1->Layers->get_Item(TVariant(7));
var_Layer1->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
var_Layer1->RotateType =
Exgaugelib_tlb::RotateTypeEnum::exRotateBilinearInterpolation;
Gauge1->set_Value(TVariant(25));
Gauge1->EndUpdate();

```

C#

```

// Change event - Occurs when the layer's value is changed.
private void exgauge1_Change(object sender,int Layer)
{

exgauge1.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.

```

```
;;0> <font ;12> <b>` + (100 - value format `0`)",exgauge1.Value,null,null));
```

```
exgauge1.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.  
-  
8",exgauge1.Layers[9].get_LayerToClientX(exgauge1.Layers[9].RotamoveCenterX,exgauge1.Layers[9].RotamoveCenterY);
```

```
exgauge1.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.  
-  
26",exgauge1.Layers[9].get_LayerToClientY(exgauge1.Layers[9].RotamoveCenterX,exgauge1.Layers[9].RotamoveCenterY);
```

```
}
```

```
//this.exgauge1.Change += new  
exontrol.EXGAUGELib.exg2antt.ChangeEventHandler(this.exgauge1_Change);
```

```
exgauge1.BeginUpdate();  
exgauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
exgauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";  
exgauge1.Layers.Count = 11;  
exgauge1.AllowSmoothChange =  
exontrol.EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;  
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[9];  
var_Layer.DefaultRotateAngle = -126;  
var_Layer.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotamove;  
var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)";  
var_Layer.ValueToRotateAngle = "(value)/100 * 360";  
var_Layer.ValueToOffsetX = "value";  
var_Layer.OffsetToValue = "value";  
var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360";  
exontrol.EXGAUGELib.Layer var_Layer1 = exgauge1.Layers[7];  
var_Layer1.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;  
var_Layer1.RotateType =  
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;  
exgauge1.Value = 25;  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="Change(Layer)" LANGUAGE="JScript">
    Gauge1.ExtraCaption("Client",0) = Gauge1.FormatABC("`<sha ;;0> <font ;12> <b>`
+ (100 - value format `0`)",Gauge1.Value,null,null);
    Gauge1.ExtraCaption("Client",4) = Gauge1.FormatABC("value -
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

    Gauge1.ExtraCaption("Client",5) = Gauge1.FormatABC("value -
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";
    Gauge1.Layers.Count = 11;
    Gauge1.AllowSmoothChange = 0;
    var var_Layer = Gauge1.Layers.Item(9);
        var_Layer.DefaultRotateAngle = -126;
        var_Layer.OnDrag = 3;
        var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)";
        var_Layer.ValueToRotateAngle = "(value)/100 * 360";
        var_Layer.ValueToOffsetX = "value";
        var_Layer.OffsetToValue = "value";
        var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360";
    var var_Layer1 = Gauge1.Layers.Item(7);
        var_Layer1.OnDrag = 2;
        var_Layer1.RotateType = 2;
```



```

Gauge1.Value = 25;
Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_Change(Layer)
    With Gauge1
        .ExtraCaption("Client",0) = .FormatABC("`<sha ;;0><font ;12><b>` + (100 - value
format `0`)",Gauge1.Value)
        .ExtraCaption("Client",4) = .FormatABC("value -
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

        .ExtraCaption("Client",5) = .FormatABC("value -
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = "`Layer` + str(value + 1) + `.png`"
        .Layers.Count = 11
        .AllowSmoothChange = 0
    End With
End Function

```

```

With .Layers.Item(9)
    .DefaultRotateAngle = -126
    .OnDrag = 3
    .RotateAngleToValue = "100 - (value / 360 * 100)"
    .ValueToRotateAngle = "(value)/100 * 360"
    .ValueToOffsetX = "value"
    .OffsetToValue = "value"
    .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
End With
With .Layers.Item(7)
    .OnDrag = 2
    .RotateType = 2
End With
.Value = 25
.EndUpdate
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

// Change event - Occurs when the layer's value is changed.
private void axGauge1_Change(object sender,
AxEXGAUGELib._IGaugeEvents_ChangeEvent e)
{

axGauge1.set_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerC
;;0> <font ;12> <b>` + (100 - value format `0`)",axGauge1.Value,null,null));

axGauge1.set_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerC
-
8",axGauge1.Layers[9].get_LayerToClientX(axGauge1.Layers[9].RotamoveCenterX,axGau

axGauge1.set_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerC

```

```

-
26",axGauge1.Layers[9].get_LayerToClientY(axGauge1.Layers[9].RotamoveCenterX,axGa
}
//this.axGauge1.Change += new
AxEXGAUGELib._IGaugeEvents_ChangeEventHandler(this.axGauge1_Change);

axGauge1.BeginUpdate();
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";
axGauge1.Layers.Count = 11;
axGauge1.AllowSmoothChange =
EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;
EXGAUGELib.Layer var_Layer = axGauge1.Layers[9];
    var_Layer.DefaultRotateAngle = -126;
    var_Layer.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotamove;
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)";
    var_Layer.ValueToRotateAngle = "(value)/100 * 360";
    var_Layer.ValueToOffsetX = "value";
    var_Layer.OffsetToValue = "value";
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360";
EXGAUGELib.Layer var_Layer1 = axGauge1.Layers[7];
    var_Layer1.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.RotateType =
EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
axGauge1.Value = 25;
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

// Change event - Occurs when the layer's value is changed.
void onEvent_Change(int _Layer)
{
    ;
    exgauge1.ExtraCaption("Client",0/*exLayerCaption*/,exgauge1.FormatABC("`<sha

```

```
;;0> <font ;12> <b>` + (100 - value format `0`)",exgauge1.Value());
```

```
exgauge1.ExtraCaption("Client",4/*exLayerCaptionLeft*/,exgauge1.FormatABC("value  
-  
8",exgauge1.Layers().Item(COMVariant::createFromInt(9)).LayerToClientX(exgauge1.Lay
```

```
exgauge1.ExtraCaption("Client",5/*exLayerCaptionTop*/,exgauge1.FormatABC("value  
-  
26",exgauge1.Layers().Item(COMVariant::createFromInt(9)).LayerToClientY(exgauge1.Lay
```

```
}
```

```
public void init()
```

```
{
```

```
    COM com_Layer,com_Layer1;
```

```
    anytype var_Layer,var_Layer1;
```

```
    ;
```

```
    super();
```

```
    exgauge1.BeginUpdate();
```

```
    exgauge1.PicturesPath("C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
```

```
    exgauge1.PicturesName("`Layer` + str(value + 1) + `.png`");
```

```
    exgauge1.Layers().Count(11);
```

```
    exgauge1.AllowSmoothChange(0/*exSmoothChangeless*/);
```

```
    var_Layer =
```

```
    COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(9));
```

```
    com_Layer = var_Layer;
```

```
        com_Layer.DefaultRotateAngle(-126);
```

```
        com_Layer.OnDrag(3/*exDoRotamove*/);
```

```
        com_Layer.RotateAngleToValue("100 - (value / 360 * 100)");
```

```
        com_Layer.ValueToRotateAngle("(value)/100 * 360");
```

```
        com_Layer.ValueToOffsetX("value");
```

```
        com_Layer.OffsetToValue("value");
```

```
        com_Layer.RotateAngleValid("int(value / 360 * 100)/100 * 360");
```

```

var_Layer1 =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(7));
com_Layer1 = var_Layer1;
    com_Layer1.OnDrag(2/*exDoRotate**/);
    com_Layer1.RotateType(2/*exRotateBilinearInterpolation**/);
exgauge1.Value(COMVariant::createFromInt(25));
exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.AxGauge1_Change(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_ChangeEvent);
begin
    with AxGauge1 do
    begin

set_ExtraCaption('Client',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,Form
;;0> <font ;12> <b>` + (100 - value format `0`)',AxGauge1.Value,Nil,Nil));

set_ExtraCaption('Client',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft,I
-
8',TObject(AxGauge1.Layers.Item[TObject(9)].LayerToClientX[TObject(AxGauge1.Layers.I

set_ExtraCaption('Client',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop,F
-
26',TObject(AxGauge1.Layers.Item[TObject(9)].LayerToClientY[TObject(AxGauge1.Layers

    end
end;

with AxGauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\Program

```

```

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
PicturesName := 'Layer` + str(value + 1) + `.png`;
Layers.Count := 11;
AllowSmoothChange := EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;
with Layers.Item[TObject(9)] do
begin
    DefaultRotateAngle := -126;
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotamove;
    RotateAngleToValue := '100 - (value / 360 * 100)';
    ValueToRotateAngle := '(value)/100 * 360';
    ValueToOffsetX := 'value';
    OffsetToValue := 'value';
    RotateAngleValid := 'int(value / 360 * 100)/100 * 360';
end;
with Layers.Item[TObject(7)] do
begin
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
    RotateType := EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
end;
Value := TObject(25);
EndUpdate();
end

```

## Delphi (standard)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.Gauge1Change(ASender: TObject; Layer : Integer);
begin
    with Gauge1 do
    begin
        ExtraCaption['Client',EXGAUGELib_TLB.exLayerCaption] := FormatABC('<sha ;0>
<font ;12> <b>` + (100 - value format `0`)',Gauge1.Value,Null,Null);
        ExtraCaption['Client',EXGAUGELib_TLB.exLayerCaptionLeft] := FormatABC('value
-
8',OleVariant(Gauge1.Layers.Item[OleVariant(9)].LayerToClientX[OleVariant(Gauge1.Laye

        ExtraCaption['Client',EXGAUGELib_TLB.exLayerCaptionTop] := FormatABC('value

```

```

-
26',OleVariant(Gauge1.Layers.Item[OleVariant(9)].LayerToClientY[OleVariant(Gauge1.Lay

    end
end;

with Gauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + str(value + 1) + `.png`;
    Layers.Count := 11;
    AllowSmoothChange := EXGAUGELib_TLB.exSmoothChangeless;
    with Layers.Item[OleVariant(9)] do
    begin
        DefaultRotateAngle := -126;
        OnDrag := EXGAUGELib_TLB.exDoRotamove;
        RotateAngleToValue := '100 - (value / 360 * 100)';
        ValueToRotateAngle := '(value)/100 * 360';
        ValueToOffsetX := 'value';
        OffsetToValue := 'value';
        RotateAngleValid := 'int(value / 360 * 100)/100 * 360';
    end;
    with Layers.Item[OleVariant(7)] do
    begin
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        RotateType := EXGAUGELib_TLB.exRotateBilinearInterpolation;
    end;
    Value := OleVariant(25);
    EndUpdate();
end

```

## VFP

*\*\*\* Change event - Occurs when the layer's value is changed. \*\*\**

LPARAMETERS Layer

```

with thisform.Gauge1
    .Object.ExtraCaption("Client",0) = .FormatABC("`<sha ;;0> <font ;12> <b>` + (100
- value format `0`)",thisform.Gauge1.Value)
    .Object.ExtraCaption("Client",4) = .FormatABC("value -
8",thisform.Gauge1.Layers.Item(9).LayerToClientX(thisform.Gauge1.Layers.Item(9).Rotan

    .Object.ExtraCaption("Client",5) = .FormatABC("value -
26",thisform.Gauge1.Layers.Item(9).LayerToClientY(thisform.Gauge1.Layers.Item(9).Rota

endwith

```

```

with thisform.Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + str(value + 1) + `.png`"
    .Layers.Count = 11
    .AllowSmoothChange = 0
    with .Layers.Item(9)
        .DefaultRotateAngle = -126
        .OnDrag = 3
        .RotateAngleToValue = "100 - (value / 360 * 100)"
        .ValueToRotateAngle = "(value)/100 * 360"
        .ValueToOffsetX = "value"
        .OffsetToValue = "value"
        .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
    endwith
    with .Layers.Item(7)
        .OnDrag = 2
        .RotateType = 2
    endwith
    .Value = 25
    .EndUpdate
endwith

```



```

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    Change = class::nativeObject_Change
endwith
*/
// Occurs when the layer's value is changed.
function nativeObject_Change(Layer)
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    oGauge.Template = [ExtraCaption("Client",0) = FormatABC("`<sha ;;0><font ;12><b>` + (100 - value format `0`)",Me.Value)] // oGauge.ExtraCaption("Client",0) =
oGauge.FormatABC("`<sha ;;0><font ;12><b>` + (100 - value format
`0`)",oGauge.Value)
    oGauge.Template = [ExtraCaption("Client",4) = FormatABC("value -
8",Me.Layers.Item(9).LayerToClientX(Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX)
// oGauge.ExtraCaption("Client",4) = oGauge.FormatABC("value -
8",oGauge.Layers.Item(9).LayerToClientX(oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX)

    oGauge.Template = [ExtraCaption("Client",5) = FormatABC("value -
26",Me.Layers.Item(9).LayerToClientY(Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX)
// oGauge.ExtraCaption("Client",5) = oGauge.FormatABC("value -
26",oGauge.Layers.Item(9).LayerToClientY(oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX)

return

local oGauge,var_Layer,var_Layer1

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.AllowSmoothChange = 0
var_Layer = oGauge.Layers.Item(9)
var_Layer.DefaultRotateAngle = -126
var_Layer.OnDrag = 3
var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)"

```



```

oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.AllowSmoothChange = 0
var_Layer = oGauge.Layers.Item(9)
    var_Layer.DefaultRotateAngle = -126
    var_Layer.OnDrag = 3
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)"
    var_Layer.ValueToRotateAngle = "(value)/100 * 360"
    var_Layer.ValueToOffsetX = "value"
    var_Layer.OffsetToValue = "value"
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360"
var_Layer1 = oGauge.Layers.Item(7)
    var_Layer1.OnDrag = 2
    var_Layer1.RotateType = 2
oGauge.Value = 25
oGauge.EndUpdate()

```

## Visual Objects

```

METHOD OCX_Exontrol1Change(Layer) CLASS MainDialog
    // Change event - Occurs when the layer's value is changed.
    oDCOCX_Exontrol1:[ExtraCaption,"Client",exLayerCaption] :=
oDCOCX_Exontrol1:FormatABC("`<sha ;;0> <font ;12> <b>` + (100 - value format
`0`)",oDCOCX_Exontrol1:Value,nil,nil)
    oDCOCX_Exontrol1:[ExtraCaption,"Client",exLayerCaptionLeft] :=
oDCOCX_Exontrol1:FormatABC("value - 8",oDCOCX_Exontrol1:Layers:[Item,9]:
[LayerToClientX,oDCOCX_Exontrol1:Layers:
[Item,9]:RotamoveCenterX,oDCOCX_Exontrol1:Layers:
[Item,9]:RotamoveCenterY],nil,nil)
    oDCOCX_Exontrol1:[ExtraCaption,"Client",exLayerCaptionTop] :=
oDCOCX_Exontrol1:FormatABC("value - 26",oDCOCX_Exontrol1:Layers:[Item,9]:
[LayerToClientY,oDCOCX_Exontrol1:Layers:
[Item,9]:RotamoveCenterX,oDCOCX_Exontrol1:Layers:

```

```
[Item,9]:RotamoveCenterY],nil,nil)
```

```
RETURN NIL
```

```
local var_Layer,var_Layer1 as ILayer
```

```
oDCOCX_Exontrol1:BeginUpdate()
```

```
oDCOCX_Exontrol1:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
oDCOCX_Exontrol1:PicturesName := "`Layer` + str(value + 1) + `.png`"
```

```
oDCOCX_Exontrol1:Layers:Count := 11
```

```
oDCOCX_Exontrol1:AllowSmoothChange := exSmoothChangeless
```

```
var_Layer := oDCOCX_Exontrol1:Layers:[Item,9]
```

```
var_Layer:DefaultRotateAngle := -126
```

```
var_Layer:OnDrag := exDoRotamove
```

```
var_Layer:RotateAngleToValue := "100 - (value / 360 * 100)"
```

```
var_Layer:ValueToRotateAngle := "(value)/100 * 360"
```

```
var_Layer:ValueToOffsetX := "value"
```

```
var_Layer:OffsetToValue := "value"
```

```
var_Layer:RotateAngleValid := "int(value / 360 * 100)/100 * 360"
```

```
var_Layer1 := oDCOCX_Exontrol1:Layers:[Item,7]
```

```
var_Layer1:OnDrag := exDoRotate
```

```
var_Layer1:RotateType := exRotateBilinearInterpolation
```

```
oDCOCX_Exontrol1:Value := 25
```

```
oDCOCX_Exontrol1:EndUpdate()
```

## PowerBuilder

```
/*begin event Change(long Layer) - Occurs when the layer's value is changed.*/  
/*
```

```
oGauge = ole_1.Object
```

```
oGauge.ExtraCaption("Client",0,oGauge.FormatABC("`<sha ;;0><font ;12><b>` +  
(100 - value format `0`)",oGauge.Value))
```

```
oGauge.ExtraCaption("Client",4,oGauge.FormatABC("value -  
8",oGauge.Layers.Item(9).LayerToClientX(oGauge.Layers.Item(9).RotamoveCenterX,oGau
```

```
oGauge.ExtraCaption("Client",5,oGauge.FormatABC("value -
```

```
26",oGauge.Layers.Item(9).LayerToClientY(oGauge.Layers.Item(9).RotamoveCenterX,oGa
```

```
*/  
/*end event Change*/
```

```
OleObject oGauge,var_Layer,var_Layer1
```

```
oGauge = ole_1.Object  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"  
oGauge.Layers.Count = 11  
oGauge.AllowSmoothChange = 0  
var_Layer = oGauge.Layers.Item(9)  
    var_Layer.DefaultRotateAngle = -126  
    var_Layer.OnDrag = 3  
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)"  
    var_Layer.ValueToRotateAngle = "(value)/100 * 360"  
    var_Layer.ValueToOffsetX = "value"  
    var_Layer.OffsetToValue = "value"  
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360"  
var_Layer1 = oGauge.Layers.Item(7)  
    var_Layer1.OnDrag = 2  
    var_Layer1.RotateType = 2  
oGauge.Value = 25  
oGauge.EndUpdate()
```

## Visual DataFlex

```
// Occurs when the layer's value is changed.
```

```
Procedure OnComChange Integer  IILayer
```

```
    Forward Send OnComChange IILayer
```

```
    Variant vA
```

```
        Get ComValue to vA
```

```
        Set ComExtraCaption "Client" OLEexLayerCaption to (ComFormatABC(Self,"<sha
```

;;0> <font ;12> <b>` + (100 - value format `0`)",vA,Nothing,Nothing))

Variant vA1

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComItem of hoLayers 9 to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Variant voLayer1

Get ComItem of hoLayer 9 to voLayer1

Handle hoLayer1

Get Create (RefClass(cComLayer)) to hoLayer1

Set pvComObject of hoLayer1 to voLayer1

Variant vX

Variant voLayers1

Get ComLayers to voLayers1

Handle hoLayers1

Get Create (RefClass(cComLayers)) to hoLayers1

Set pvComObject of hoLayers1 to voLayers1

Variant voLayer2

Get ComItem of hoLayers1 9 to voLayer2

Handle hoLayer2

Get Create (RefClass(cComLayer)) to hoLayer2

Set pvComObject of hoLayer2 to voLayer2

Variant voLayer3

Get ComItem of hoLayer2 9 to voLayer3

Handle hoLayer3

Get Create (RefClass(cComLayer)) to hoLayer3

Set pvComObject of hoLayer3 to voLayer3

Get ComRotamoveCenterX of hoLayer3 to vX

Send Destroy to hoLayer3

Send Destroy to hoLayer2

Send Destroy to hoLayers1

Variant vY

Variant voLayers2

Get ComLayers to voLayers2

Handle hoLayers2

Get Create (RefClass(cComLayers)) to hoLayers2

Set pvComObject of hoLayers2 to voLayers2

Variant voLayer4

Get ComItem of hoLayers2 9 to voLayer4

Handle hoLayer4

Get Create (RefClass(cComLayer)) to hoLayer4

Set pvComObject of hoLayer4 to voLayer4

Variant voLayer5

Get ComItem of hoLayer4 9 to voLayer5

Handle hoLayer5

Get Create (RefClass(cComLayer)) to hoLayer5

Set pvComObject of hoLayer5 to voLayer5

Get ComRotamoveCenterY of hoLayer5 to vY

Send Destroy to hoLayer5

Send Destroy to hoLayer4

Send Destroy to hoLayers2

Get ComLayerToClientX of hoLayer1 vX vY to vA1

Send Destroy to hoLayer1

Send Destroy to hoLayer

Send Destroy to hoLayers

Set ComExtraCaption "Client" OLEexLayerCaptionLeft to

(ComFormatABC(Self,"value - 8",vA1,Nothing,Nothing))

Variant vA2

Variant voLayers3

Get ComLayers to voLayers3

Handle hoLayers3

Get Create (RefClass(cComLayers)) to hoLayers3

Set pvComObject of hoLayers3 to voLayers3

Variant voLayer6

Get ComItem of hoLayers3 9 to voLayer6

Handle hoLayer6

Get Create (RefClass(cComLayer)) to hoLayer6

Set pvComObject of hoLayer6 to voLayer6

Variant voLayer7

Get ComItem of hoLayer6 9 to voLayer7

Handle hoLayer7

Get Create (RefClass(cComLayer)) to hoLayer7

Set pvComObject of hoLayer7 to voLayer7

Variant vX1

Variant voLayers4

Get ComLayers to voLayers4

Handle hoLayers4

Get Create (RefClass(cComLayers)) to hoLayers4

Set pvComObject of hoLayers4 to voLayers4

Variant voLayer8

Get ComItem of hoLayers4 9 to voLayer8

Handle hoLayer8

Get Create (RefClass(cComLayer)) to hoLayer8

Set pvComObject of hoLayer8 to voLayer8

Variant voLayer9

Get ComItem of hoLayer8 9 to voLayer9

Handle hoLayer9

Get Create (RefClass(cComLayer)) to hoLayer9

Set pvComObject of hoLayer9 to voLayer9

Get ComRotamoveCenterX of hoLayer9 to vX1

Send Destroy to hoLayer9

Send Destroy to hoLayer8

Send Destroy to hoLayers4

Variant vY1

Variant voLayers5

Get ComLayers to voLayers5

Handle hoLayers5

Get Create (RefClass(cComLayers)) to hoLayers5

Set pvComObject of hoLayers5 to voLayers5

Variant voLayer10

Get ComItem of hoLayers5 9 to voLayer10

Handle hoLayer10

Get Create (RefClass(cComLayer)) to hoLayer10

Set pvComObject of hoLayer10 to voLayer10

Variant voLayer11



```

        Get ComItem of hoLayer10 9 to voLayer11
        Handle hoLayer11
        Get Create (RefClass(cComLayer)) to hoLayer11
        Set pvComObject of hoLayer11 to voLayer11
            Get ComRotamoveCenterY of hoLayer11 to vY1
        Send Destroy to hoLayer11
    Send Destroy to hoLayer10
    Send Destroy to hoLayers5
    Get ComLayerToClientY of hoLayer7 vX1 vY1 to vA2
    Send Destroy to hoLayer7
    Send Destroy to hoLayer6
    Send Destroy to hoLayers3
    Set ComExtraCaption "Client" OLEexLayerCaptionTop to
(ComFormatABC(Self,"value - 26",vA2,Nothing,Nothing))
End_Procedure

```

#### Procedure OnCreate

```

    Forward Send OnCreate
    Send ComBeginUpdate
    Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    Set ComPicturesName to "`Layer` + str(value + 1) + `.png`"
    Variant voLayers6
    Get ComLayers to voLayers6
    Handle hoLayers6
    Get Create (RefClass(cComLayers)) to hoLayers6
    Set pvComObject of hoLayers6 to voLayers6
        Set ComCount of hoLayers6 to 11
    Send Destroy to hoLayers6
    Set ComAllowSmoothChange to OLEexSmoothChangeless
    Variant voLayers7
    Get ComLayers to voLayers7
    Handle hoLayers7
    Get Create (RefClass(cComLayers)) to hoLayers7
    Set pvComObject of hoLayers7 to voLayers7
        Variant voLayer12
        Get ComItem of hoLayers7 9 to voLayer12

```

```

Handle hoLayer12
Get Create (RefClass(cComLayer)) to hoLayer12
Set pvComObject of hoLayer12 to voLayer12
    Set ComDefaultRotateAngle of hoLayer12 to -126
    Set ComOnDrag of hoLayer12 to OLEexDoRotamove
    Set ComRotateAngleToValue of hoLayer12 to "100 - (value / 360 * 100)"
    Set ComValueToRotateAngle of hoLayer12 to "(value)/100 * 360"
    Set ComValueToOffsetX of hoLayer12 to "value"
    Set ComOffsetToValue of hoLayer12 to "value"
    Set ComRotateAngleValid of hoLayer12 to "int(value / 360 * 100)/100 * 360"
Send Destroy to hoLayer12
Send Destroy to hoLayers7
Variant voLayers8
Get ComLayers to voLayers8
Handle hoLayers8
Get Create (RefClass(cComLayers)) to hoLayers8
Set pvComObject of hoLayers8 to voLayers8
    Variant voLayer13
    Get ComItem of hoLayers8 7 to voLayer13
    Handle hoLayer13
    Get Create (RefClass(cComLayer)) to hoLayer13
    Set pvComObject of hoLayer13 to voLayer13
        Set ComOnDrag of hoLayer13 to OLEexDoRotate
        Set ComRotateType of hoLayer13 to OLEexRotateBilinearInterpolation
    Send Destroy to hoLayer13
Send Destroy to hoLayers8
Set ComValue to 25
Send ComEndUpdate
End_Procedure

```

## XBase++

```

PROCEDURE OnChange(oGauge,Layer)

```

```

oGauge:SetProperty("ExtraCaption","Client",0/*exLayerCaption*/,oGauge:FormatABC("
;;0> <font ;12> <b>` + (100 - value format `0`)",oGauge:Value()))

```

```

oGauge:SetProperty("ExtraCaption","Client",4/*exLayerCaptionLeft*/,oGauge:FormatA
-
8",oGauge:Layers:Item(9):LayerToClientX(oGauge:Layers:Item(9):RotamoveCenterX(),oG
oGauge:SetProperty("ExtraCaption","Client",5/*exLayerCaptionTop*/,oGauge:FormatA
-
26",oGauge:Layers:Item(9):LayerToClientY(oGauge:Layers:Item(9):RotamoveCenterX(),oG

```

```

RETURN

```

```

#include "AppEvent.ch"
#include "ActiveX.ch"

```

```

PROCEDURE Main

```

```

    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge
    LOCAL oLayer,oLayer1

```

```

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

```

```

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

```

```

        oGauge:Change := {|| Layer| OnChange(oGauge,Layer)} /*Occurs when the layer's
value is changed.*/

```

```

        oGauge:BeginUpdate()
        oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        oGauge:PicturesName := ""Layer` + str(value + 1) + `.png""

```

```
oGauge:Layers():Count := 11
oGauge:AllowSmoothChange := 0/*exSmoothChangeless*/
oLayer := oGauge:Layers:Item(9)
  oLayer:DefaultRotateAngle := -126
  oLayer:OnDrag := 3/*exDoRotamove*/
  oLayer:RotateAngleToValue := "100 - (value / 360 * 100)"
  oLayer:ValueToRotateAngle := "(value)/100 * 360"
  oLayer:ValueToOffsetX := "value"
  oLayer:OffsetToValue := "value"
  oLayer:RotateAngleValid := "int(value / 360 * 100)/100 * 360"
oLayer1 := oGauge:Layers:Item(7)
  oLayer1:OnDrag := 2/*exDoRotate*/
  oLayer1:RotateType := 2/*exRotateBilinearInterpolation*/
oGauge:Value := 25
oGauge:EndUpdate()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

## property Layer.LayerToClientY (X as Variant, Y as Variant) as Long

Converts the x-position of the layer to control's client x-position.

Type	Description
X as Variant	A Lone expression that specifies the x-position of the point within the layer
Y as Variant	A Lone expression that specifies the y-position of the point within the layer
Long	A Long expression that specifies the y-position of the point on the control's view that's equivalent of the point on the layer.

The [LayerToClientX](#) / LayerToClientY converts the (x,y)-point on the layer to control's view point. The [LayerToClientX](#) / LayerToClientY properties translate a point from the layer ( as it is moved or rotated ) to the control's view. For instance, you can display the current value of the control on the knob you are rotating. The [RotamoveCenterX](#) / [RotamoveCenterY](#) specifies the (x,y)-position of the layer's center, while the layer's drag operation is `exDoRotamove`. The [OnDrag](#) property indicates the action to be performed when the user drags the layer.

Any of the following properties can be used to display a HTML caption:

- [Caption](#) property specifies the caption to be shown on the control's foreground.
- [ExtraCaption](#) property specifies any extra caption to be shown on the control's foreground.
- [Foreground.Caption](#) specifies the caption to be shown on the layer's foreground.
- [Foreground.ExtraCaption](#) specifies any extra caption to be shown on the layer's foreground.



The following sample shows how you can use the `LayerToClientX` / [LayerToClientY](#) properties to display the layer's value on the knob:

### VBA (MS Access, Excell...)

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)
```

```
    With Gauge1
```

```
        .ExtraCaption("Client",0) = .FormatABC("`<sha ;;0><font ;12><b>` + (100 - value  
format `0`)",Gauge1.Value)
```

```
        .ExtraCaption("Client",4) = .FormatABC("value -  
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gau
```

```
        .ExtraCaption("Client",5) = .FormatABC("value -  
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau
```

```
    End With
```

```
End Sub
```

```
With Gauge1
```

```
    .BeginUpdate
```

```
    .PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```

.PicturesName = "`Layer` + str(value + 1) + `.png`"
.Layers.Count = 11
.AllowSmoothChange = 0
With .Layers.Item(9)
    .DefaultRotateAngle = -126
    .OnDrag = 3
    .RotateAngleToValue = "100 - (value / 360 * 100)"
    .ValueToRotateAngle = "(value)/100 * 360"
    .ValueToOffsetX = "value"
    .OffsetToValue = "value"
    .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
End With
With .Layers.Item(7)
    .OnDrag = 2
    .RotateType = 2
End With
.Value = 25
.EndUpdate
End With

```

## VB6

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)
```

```
    With Gauge1
```

```
        .ExtraCaption("Client",exLayerCaption) = .FormatABC("`<sha ;;0> <font ;12> <b>`  
+ (100 - value format `0`)",Gauge1.Value)
```

```
        .ExtraCaption("Client",exLayerCaptionLeft) = .FormatABC("value -  
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gaug
```

```
        .ExtraCaption("Client",exLayerCaptionTop) = .FormatABC("value -  
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau
```

```
    End With
```

```
End Sub
```

```
With Gauge1
```

```

.BeginUpdate
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + str(value + 1) + `.png`"
.Layers.Count = 11
.AllowSmoothChange = exSmoothChangeless
With .Layers.Item(9)
.DefaultRotateAngle = -126
.OnDrag = exDoRotamove
.RotateAngleToValue = "100 - (value / 360 * 100)"
.ValueToRotateAngle = "(value)/100 * 360"
.ValueToOffsetX = "value"
.OffsetToValue = "value"
.RotateAngleValid = "int(value / 360 * 100)/100 * 360"
End With
With .Layers.Item(7)
.OnDrag = exDoRotate
.RotateType = exRotateBilinearInterpolation
End With
.Value = 25
.EndUpdate
End With

```

## VB.NET

*' Change event - Occurs when the layer's value is changed.*

```

Private Sub Exgauge1_Change(ByVal sender As System.Object,ByVal Layer As Integer)
Handles Exgauge1.Change
    With Exgauge1

.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
;;0> <font ;12> <b>` + (100 - value format `0`)",Exgauge1.Value))

.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
-
8",Exgauge1.Layers.Item(9).get_LayerToClientX(Exgauge1.Layers.Item(9).RotamoveCent

```



```

.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCa
-
26",Exgauge1.Layers.Item(9).get_LayerToClientY(Exgauge1.Layers.Item(9).RotamoveCen

    End With
End Sub

With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + str(value + 1) + `.png`"
    .Layers.Count = 11
    .AllowSmoothChange =
exontrol.EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless
    With .Layers.Item(9)
        .DefaultRotateAngle = -126
        .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotamove
        .RotateAngleToValue = "100 - (value / 360 * 100)"
        .ValueToRotateAngle = "(value)/100 * 360"
        .ValueToOffsetX = "value"
        .OffsetToValue = "value"
        .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
    End With
    With .Layers.Item(7)
        .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
        .RotateType =
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
    End With
    .Value = 25
    .EndUpdate()
End With

```

## VB.NET for /COM

*' Change event - Occurs when the layer's value is changed.*

Private Sub AxGauge1\_Change(ByVal sender As System.Object, ByVal e As AxEXGAUGELib.\_IGaugeEvents\_ChangeEvent) Handles AxGauge1.Change  
With AxGauge1

.set\_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,.Fo  
;;0> <font ;12> <b>` + (100 - value format `0`)",AxGauge1.Value))

.set\_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft  
-  
8",AxGauge1.Layers.Item(9).LayerToClientX(AxGauge1.Layers.Item(9).RotamoveCenterX,

.set\_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop  
-  
26",AxGauge1.Layers.Item(9).LayerToClientY(AxGauge1.Layers.Item(9).RotamoveCenter,

End With  
End Sub

With AxGauge1

.BeginUpdate()  
.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
.PicturesName = "`Layer` + str(value + 1) + `.png`"  
.Layers.Count = 11  
.AllowSmoothChange = EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless  
With .Layers.Item(9)  
.DefaultRotateAngle = -126  
.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotamove  
.RotateAngleToValue = "100 - (value / 360 \* 100)"  
.ValueToRotateAngle = "(value)/100 \* 360"  
.ValueToOffsetX = "value"  
.OffsetToValue = "value"  
.RotateAngleValid = "int(value / 360 \* 100)/100 \* 360"

End With

With .Layers.Item(7)

.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate

```

.RotateType = EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation
End With
.Value = 25
.EndUpdate()
End With

```

## C++

```

// Change event - Occurs when the layer's value is changed.
void OnChangeGauge1(long Layer)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
        Library'
        #import <ExGauge.dll>
        using namespace EXGAUGELib;
    */
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    spGauge1->PutExtraCaption("Client",EXGAUGELib::exLayerCaption,spGauge1-
> FormatABC(L"<sha ;;0> <font ;12> <b>` + (100 - value format `0`)",spGauge1-
> GetValue(),vtMissing,vtMissing));
    spGauge1->PutExtraCaption("Client",EXGAUGELib::exLayerCaptionLeft,spGauge1-
> FormatABC(L"value - 8",spGauge1->GetLayers()->GetItem(long(9))-
> GetLayerToClientX(spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterX(),spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterY()),vtMissing,vtMissing));
    spGauge1->PutExtraCaption("Client",EXGAUGELib::exLayerCaptionTop,spGauge1-
> FormatABC(L"value - 26",spGauge1->GetLayers()->GetItem(long(9))-
> GetLayerToClientY(spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterX(),spGauge1->GetLayers()->GetItem(long(9))-
> GetRotamoveCenterY()),vtMissing,vtMissing));
}

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();

```

```

spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + str(value + 1) + `.png`");
spGauge1->GetLayers()->PutCount(11);
spGauge1->PutAllowSmoothChange(EXGAUGELib::exSmoothChangeless);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(long(9));
    var_Layer->PutDefaultRotateAngle(-126);
    var_Layer->PutOnDrag(EXGAUGELib::exDoRotamove);
    var_Layer->PutRotateAngleToValue(L"100 - (value / 360 * 100)");
    var_Layer->PutValueToRotateAngle(L"(value)/100 * 360");
    var_Layer->PutValueToOffsetX(L"value");
    var_Layer->PutOffsetToValue(L"value");
    var_Layer->PutRotateAngleValid(L"int(value / 360 * 100)/100 * 360");
EXGAUGELib::ILayerPtr var_Layer1 = spGauge1->GetLayers()->GetItem(long(7));
    var_Layer1->PutOnDrag(EXGAUGELib::exDoRotate);
    var_Layer1->PutRotateType(EXGAUGELib::exRotateBilinearInterpolation);
spGauge1->PutValue(long(25));
spGauge1->EndUpdate();

```

## C++ Builder

```

// Change event - Occurs when the layer's value is changed.
void __fastcall TForm1::Gauge1Change(TObject *Sender,long Layer)
{
    Gauge1-
> ExtraCaption[TVariant("Client"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCa
    = TVariant(Gauge1->FormatABC(L"<sha ;;0> <font ;12> <b>` + (100 - value format
`0`)",TVariant(Gauge1->get_Value()),TNoParam(),TNoParam()));
    Gauge1-
> ExtraCaption[TVariant("Client"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCa
    = TVariant(Gauge1->FormatABC(L"value - 8",TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->get_LayerToClientX(TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterX),TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterY))),TNoParam(),TNoParam()));
    Gauge1-

```

```

> ExtraCaption[TVariant("Client"),Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCa
= TVariant(Gauge1->FormatABC(L"value - 26",TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->get_LayerToClientY(TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterX),TVariant(Gauge1->Layers-
>get_Item(TVariant(9))->RotamoveCenterY))),TNoParam(),TNoParam()));
}

Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + str(value + 1) + `.png`;
Gauge1->Layers->Count = 11;
Gauge1->AllowSmoothChange =
Exgaugelib_tlb::SmoothPropertyEnum::exSmoothChangeless;
Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(9));
var_Layer->DefaultRotateAngle = -126;
var_Layer->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotamove;
var_Layer->RotateAngleToValue = L"100 - (value / 360 * 100)";
var_Layer->ValueToRotateAngle = L"(value)/100 * 360";
var_Layer->ValueToOffsetX = L"value";
var_Layer->OffsetToValue = L"value";
var_Layer->RotateAngleValid = L"int(value / 360 * 100)/100 * 360";
Exgaugelib_tlb::ILayerPtr var_Layer1 = Gauge1->Layers->get_Item(TVariant(7));
var_Layer1->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoRotate;
var_Layer1->RotateType =
Exgaugelib_tlb::RotateTypeEnum::exRotateBilinearInterpolation;
Gauge1->set_Value(TVariant(25));
Gauge1->EndUpdate();

```

C#

```

// Change event - Occurs when the layer's value is changed.
private void exgauge1_Change(object sender,int Layer)
{

exgauge1.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.

```

```
;;0> <font ;12> <b>` + (100 - value format `0`)",exgauge1.Value,null,null));
```

```
exgauge1.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.  
-  
8",exgauge1.Layers[9].get_LayerToClientX(exgauge1.Layers[9].RotamoveCenterX,exgauge1.Layers[9].RotamoveCenterY);
```

```
exgauge1.set_ExtraCaption("Client",exontrol.EXGAUGELib.PropertyLayerCaptionEnum.  
-  
26",exgauge1.Layers[9].get_LayerToClientY(exgauge1.Layers[9].RotamoveCenterX,exgauge1.Layers[9].RotamoveCenterY);
```

```
}
```

```
//this.exgauge1.Change += new  
exontrol.EXGAUGELib.exg2antt.ChangeEventHandler(this.exgauge1_Change);
```

```
exgauge1.BeginUpdate();  
exgauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
exgauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";  
exgauge1.Layers.Count = 11;  
exgauge1.AllowSmoothChange =  
exontrol.EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;  
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[9];  
    var_Layer.DefaultRotateAngle = -126;  
    var_Layer.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotamove;  
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)";  
    var_Layer.ValueToRotateAngle = "(value)/100 * 360";  
    var_Layer.ValueToOffsetX = "value";  
    var_Layer.OffsetToValue = "value";  
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360";  
exontrol.EXGAUGELib.Layer var_Layer1 = exgauge1.Layers[7];  
    var_Layer1.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;  
    var_Layer1.RotateType =  
exontrol.EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;  
exgauge1.Value = 25;  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="Change(Layer)" LANGUAGE="JScript">
    Gauge1.ExtraCaption("Client",0) = Gauge1.FormatABC("`<sha ;;0> <font ;12> <b>`
+ (100 - value format `0`)",Gauge1.Value,null,null);
    Gauge1.ExtraCaption("Client",4) = Gauge1.FormatABC("value -
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

    Gauge1.ExtraCaption("Client",5) = Gauge1.FormatABC("value -
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";
    Gauge1.Layers.Count = 11;
    Gauge1.AllowSmoothChange = 0;
    var var_Layer = Gauge1.Layers.Item(9);
        var_Layer.DefaultRotateAngle = -126;
        var_Layer.OnDrag = 3;
        var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)";
        var_Layer.ValueToRotateAngle = "(value)/100 * 360";
        var_Layer.ValueToOffsetX = "value";
        var_Layer.OffsetToValue = "value";
        var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360";
    var var_Layer1 = Gauge1.Layers.Item(7);
        var_Layer1.OnDrag = 2;
        var_Layer1.RotateType = 2;
```

```

Gauge1.Value = 25;
Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_Change(Layer)
    With Gauge1
        .ExtraCaption("Client",0) = .FormatABC("`<sha ;;0><font ;12><b>` + (100 - value
format `0`)",Gauge1.Value)
        .ExtraCaption("Client",4) = .FormatABC("value -
8",Gauge1.Layers.Item(9).LayerToClientX(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

        .ExtraCaption("Client",5) = .FormatABC("value -
26",Gauge1.Layers.Item(9).LayerToClientY(Gauge1.Layers.Item(9).RotamoveCenterX,Gau

    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = "`Layer` + str(value + 1) + `.png`"
        .Layers.Count = 11
        .AllowSmoothChange = 0
    End With
End Function

```



```

With .Layers.Item(9)
    .DefaultRotateAngle = -126
    .OnDrag = 3
    .RotateAngleToValue = "100 - (value / 360 * 100)"
    .ValueToRotateAngle = "(value)/100 * 360"
    .ValueToOffsetX = "value"
    .OffsetToValue = "value"
    .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
End With
With .Layers.Item(7)
    .OnDrag = 2
    .RotateType = 2
End With
.Value = 25
.EndUpdate
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

// Change event - Occurs when the layer's value is changed.
private void axGauge1_Change(object sender,
AxEXGAUGELib._IGaugeEvents_ChangeEvent e)
{

axGauge1.set_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerC
;;0> <font ;12> <b>` + (100 - value format `0`)",axGauge1.Value,null,null));

axGauge1.set_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerC
-
8",axGauge1.Layers[9].get_LayerToClientX(axGauge1.Layers[9].RotamoveCenterX,axGau

axGauge1.set_ExtraCaption("Client",EXGAUGELib.PropertyLayerCaptionEnum.exLayerC

```

```

-
26",axGauge1.Layers[9].get_LayerToClientY(axGauge1.Layers[9].RotamoveCenterX,axGa
}
//this.axGauge1.Change += new
AxEXGAUGELib._IGaugeEvents_ChangeEventHandler(this.axGauge1_Change);

axGauge1.BeginUpdate();
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";
axGauge1.Layers.Count = 11;
axGauge1.AllowSmoothChange =
EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;
EXGAUGELib.Layer var_Layer = axGauge1.Layers[9];
    var_Layer.DefaultRotateAngle = -126;
    var_Layer.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotamove;
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)";
    var_Layer.ValueToRotateAngle = "(value)/100 * 360";
    var_Layer.ValueToOffsetX = "value";
    var_Layer.OffsetToValue = "value";
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360";
EXGAUGELib.Layer var_Layer1 = axGauge1.Layers[7];
    var_Layer1.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.RotateType =
EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
axGauge1.Value = 25;
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

// Change event - Occurs when the layer's value is changed.
void onEvent_Change(int _Layer)
{
    ;
    exgauge1.ExtraCaption("Client",0/*exLayerCaption*/,exgauge1.FormatABC("`<sha

```

```
;;0> <font ;12> <b>` + (100 - value format `0`)",exgauge1.Value());
```

```
exgauge1.ExtraCaption("Client",4/*exLayerCaptionLeft*/,exgauge1.FormatABC("value  
-  
8",exgauge1.Layers().Item(COMVariant::createFromInt(9)).LayerToClientX(exgauge1.Lay
```

```
exgauge1.ExtraCaption("Client",5/*exLayerCaptionTop*/,exgauge1.FormatABC("value  
-  
26",exgauge1.Layers().Item(COMVariant::createFromInt(9)).LayerToClientY(exgauge1.Lay
```

```
}
```

```
public void init()
```

```
{
```

```
    COM com_Layer,com_Layer1;
```

```
    anytype var_Layer,var_Layer1;
```

```
    ;
```

```
    super();
```

```
    exgauge1.BeginUpdate();
```

```
    exgauge1.PicturesPath("C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
```

```
    exgauge1.PicturesName("`Layer` + str(value + 1) + `.png`");
```

```
    exgauge1.Layers().Count(11);
```

```
    exgauge1.AllowSmoothChange(0/*exSmoothChangeless*/);
```

```
    var_Layer =
```

```
    COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(9));
```

```
    com_Layer = var_Layer;
```

```
        com_Layer.DefaultRotateAngle(-126);
```

```
        com_Layer.OnDrag(3/*exDoRotamove*/);
```

```
        com_Layer.RotateAngleToValue("100 - (value / 360 * 100)");
```

```
        com_Layer.ValueToRotateAngle("(value)/100 * 360");
```

```
        com_Layer.ValueToOffsetX("value");
```

```
        com_Layer.OffsetToValue("value");
```

```
        com_Layer.RotateAngleValid("int(value / 360 * 100)/100 * 360");
```

```

var_Layer1 =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(7));
com_Layer1 = var_Layer1;
    com_Layer1.OnDrag(2/*exDoRotate**/);
    com_Layer1.RotateType(2/*exRotateBilinearInterpolation**/);
exgauge1.Value(COMVariant::createFromInt(25));
exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.AxGauge1_Change(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_ChangeEvent);
begin
    with AxGauge1 do
    begin

set_ExtraCaption('Client',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,Form
;;0> <font ;12> <b>` + (100 - value format `0`)',AxGauge1.Value,Nil,Nil));

set_ExtraCaption('Client',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionLeft,I
-
8',TObject(AxGauge1.Layers.Item[TObject(9)].LayerToClientX[TObject(AxGauge1.Layers.I

set_ExtraCaption('Client',EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaptionTop,F
-
26',TObject(AxGauge1.Layers.Item[TObject(9)].LayerToClientY[TObject(AxGauge1.Layers

    end
end;

with AxGauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\Program

```

```

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
PicturesName := 'Layer` + str(value + 1) + `.png`;
Layers.Count := 11;
AllowSmoothChange := EXGAUGELib.SmoothPropertyEnum.exSmoothChangeless;
with Layers.Item[TObject(9)] do
begin
    DefaultRotateAngle := -126;
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotamove;
    RotateAngleToValue := '100 - (value / 360 * 100)';
    ValueToRotateAngle := '(value)/100 * 360';
    ValueToOffsetX := 'value';
    OffsetToValue := 'value';
    RotateAngleValid := 'int(value / 360 * 100)/100 * 360';
end;
with Layers.Item[TObject(7)] do
begin
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;
    RotateType := EXGAUGELib.RotateTypeEnum.exRotateBilinearInterpolation;
end;
Value := TObject(25);
EndUpdate();
end

```

## Delphi (standard)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.Gauge1Change(ASender: TObject; Layer : Integer);
begin
    with Gauge1 do
    begin
        ExtraCaption['Client',EXGAUGELib_TLB.exLayerCaption] := FormatABC('<sha ;0>
<font ;12> <b>` + (100 - value format `0`)',Gauge1.Value,Null,Null);
        ExtraCaption['Client',EXGAUGELib_TLB.exLayerCaptionLeft] := FormatABC('value
-
8',OleVariant(Gauge1.Layers.Item[OleVariant(9)].LayerToClientX[OleVariant(Gauge1.Laye

        ExtraCaption['Client',EXGAUGELib_TLB.exLayerCaptionTop] := FormatABC('value

```

```

-
26',OleVariant(Gauge1.Layers.Item[OleVariant(9)].LayerToClientY[OleVariant(Gauge1.Lay

    end
end;

with Gauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := '\Layer` + str(value + 1) + `.png`;
    Layers.Count := 11;
    AllowSmoothChange := EXGAUGELib_TLB.exSmoothChangeless;
    with Layers.Item[OleVariant(9)] do
    begin
        DefaultRotateAngle := -126;
        OnDrag := EXGAUGELib_TLB.exDoRotamove;
        RotateAngleToValue := '100 - (value / 360 * 100)';
        ValueToRotateAngle := '(value)/100 * 360';
        ValueToOffsetX := 'value';
        OffsetToValue := 'value';
        RotateAngleValid := 'int(value / 360 * 100)/100 * 360';
    end;
    with Layers.Item[OleVariant(7)] do
    begin
        OnDrag := EXGAUGELib_TLB.exDoRotate;
        RotateType := EXGAUGELib_TLB.exRotateBilinearInterpolation;
    end;
    Value := OleVariant(25);
    EndUpdate();
end

```

## VFP

\*\*\* *Change event - Occurs when the layer's value is changed.* \*\*\*

LPARAMETERS Layer

```

with thisform.Gauge1
    .Object.ExtraCaption("Client",0) = .FormatABC("`<sha ;;0> <font ;12> <b>` + (100
- value format `0`)",thisform.Gauge1.Value)
    .Object.ExtraCaption("Client",4) = .FormatABC("value -
8",thisform.Gauge1.Layers.Item(9).LayerToClientX(thisform.Gauge1.Layers.Item(9).Rotan

    .Object.ExtraCaption("Client",5) = .FormatABC("value -
26",thisform.Gauge1.Layers.Item(9).LayerToClientY(thisform.Gauge1.Layers.Item(9).Rota

endwith

```

```

with thisform.Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + str(value + 1) + `.png`"
    .Layers.Count = 11
    .AllowSmoothChange = 0
    with .Layers.Item(9)
        .DefaultRotateAngle = -126
        .OnDrag = 3
        .RotateAngleToValue = "100 - (value / 360 * 100)"
        .ValueToRotateAngle = "(value)/100 * 360"
        .ValueToOffsetX = "value"
        .OffsetToValue = "value"
        .RotateAngleValid = "int(value / 360 * 100)/100 * 360"
    endwith
    with .Layers.Item(7)
        .OnDrag = 2
        .RotateType = 2
    endwith
    .Value = 25
    .EndUpdate
endwith

```

```

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    Change = class::nativeObject_Change
endwith
*/
// Occurs when the layer's value is changed.
function nativeObject_Change(Layer)
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    oGauge.Template = [ExtraCaption("Client",0) = FormatABC("`<sha ;;0><font ;12><b>` + (100 - value format `0`)",Me.Value)] // oGauge.ExtraCaption("Client",0) =
oGauge.FormatABC("`<sha ;;0><font ;12><b>` + (100 - value format
`0`)",oGauge.Value)
    oGauge.Template = [ExtraCaption("Client",4) = FormatABC("value -
8",Me.Layers.Item(9).LayerToClientX(Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX)
// oGauge.ExtraCaption("Client",4) = oGauge.FormatABC("value -
8",oGauge.Layers.Item(9).LayerToClientX(oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX)

    oGauge.Template = [ExtraCaption("Client",5) = FormatABC("value -
26",Me.Layers.Item(9).LayerToClientY(Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX,Me.Layers.Item(9).RotamoveCenterX)
// oGauge.ExtraCaption("Client",5) = oGauge.FormatABC("value -
26",oGauge.Layers.Item(9).LayerToClientY(oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX,oGauge.Layers.Item(9).RotamoveCenterX)

return

local oGauge,var_Layer,var_Layer1

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.AllowSmoothChange = 0
var_Layer = oGauge.Layers.Item(9)
    var_Layer.DefaultRotateAngle = -126
    var_Layer.OnDrag = 3
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)"

```





```

oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.AllowSmoothChange = 0
var_Layer = oGauge.Layers.Item(9)
    var_Layer.DefaultRotateAngle = -126
    var_Layer.OnDrag = 3
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)"
    var_Layer.ValueToRotateAngle = "(value)/100 * 360"
    var_Layer.ValueToOffsetX = "value"
    var_Layer.OffsetToValue = "value"
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360"
var_Layer1 = oGauge.Layers.Item(7)
    var_Layer1.OnDrag = 2
    var_Layer1.RotateType = 2
oGauge.Value = 25
oGauge.EndUpdate()

```

## Visual Objects

```

METHOD OCX_Exontrol1Change(Layer) CLASS MainDialog
    // Change event - Occurs when the layer's value is changed.
    oDCOCX_Exontrol1:[ExtraCaption,"Client",exLayerCaption] :=
oDCOCX_Exontrol1:FormatABC("`<sha ;;0> <font ;12> <b>` + (100 - value format
`0`)",oDCOCX_Exontrol1:Value,nil,nil)
    oDCOCX_Exontrol1:[ExtraCaption,"Client",exLayerCaptionLeft] :=
oDCOCX_Exontrol1:FormatABC("value - 8",oDCOCX_Exontrol1:Layers:[Item,9]:
[LayerToClientX,oDCOCX_Exontrol1:Layers:
[Item,9]:RotamoveCenterX,oDCOCX_Exontrol1:Layers:
[Item,9]:RotamoveCenterY],nil,nil)
    oDCOCX_Exontrol1:[ExtraCaption,"Client",exLayerCaptionTop] :=
oDCOCX_Exontrol1:FormatABC("value - 26",oDCOCX_Exontrol1:Layers:[Item,9]:
[LayerToClientY,oDCOCX_Exontrol1:Layers:
[Item,9]:RotamoveCenterX,oDCOCX_Exontrol1:Layers:

```

```
[Item,9]:RotamoveCenterY],nil,nil)
```

```
RETURN NIL
```

```
local var_Layer,var_Layer1 as ILayer
```

```
oDCOCX_Exontrol1:BeginUpdate()
```

```
oDCOCX_Exontrol1:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
oDCOCX_Exontrol1:PicturesName := "`Layer` + str(value + 1) + `.png`"
```

```
oDCOCX_Exontrol1:Layers:Count := 11
```

```
oDCOCX_Exontrol1:AllowSmoothChange := exSmoothChangeless
```

```
var_Layer := oDCOCX_Exontrol1:Layers:[Item,9]
```

```
var_Layer:DefaultRotateAngle := -126
```

```
var_Layer:OnDrag := exDoRotamove
```

```
var_Layer:RotateAngleToValue := "100 - (value / 360 * 100)"
```

```
var_Layer:ValueToRotateAngle := "(value)/100 * 360"
```

```
var_Layer:ValueToOffsetX := "value"
```

```
var_Layer:OffsetToValue := "value"
```

```
var_Layer:RotateAngleValid := "int(value / 360 * 100)/100 * 360"
```

```
var_Layer1 := oDCOCX_Exontrol1:Layers:[Item,7]
```

```
var_Layer1:OnDrag := exDoRotate
```

```
var_Layer1:RotateType := exRotateBilinearInterpolation
```

```
oDCOCX_Exontrol1:Value := 25
```

```
oDCOCX_Exontrol1:EndUpdate()
```

## PowerBuilder

```
/*begin event Change(long Layer) - Occurs when the layer's value is changed.*/  
/*
```

```
oGauge = ole_1.Object
```

```
oGauge.ExtraCaption("Client",0,oGauge.FormatABC("`<sha ;;0><font ;12><b>` +  
(100 - value format `0`)",oGauge.Value))
```

```
oGauge.ExtraCaption("Client",4,oGauge.FormatABC("value -  
8",oGauge.Layers.Item(9).LayerToClientX(oGauge.Layers.Item(9).RotamoveCenterX,oGau
```

```
oGauge.ExtraCaption("Client",5,oGauge.FormatABC("value -
```

```
26",oGauge.Layers.Item(9).LayerToClientY(oGauge.Layers.Item(9).RotamoveCenterX,oGa
```

```
*/  
/*end event Change*/
```

```
OleObject oGauge,var_Layer,var_Layer1
```

```
oGauge = ole_1.Object  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"  
oGauge.Layers.Count = 11  
oGauge.AllowSmoothChange = 0  
var_Layer = oGauge.Layers.Item(9)  
    var_Layer.DefaultRotateAngle = -126  
    var_Layer.OnDrag = 3  
    var_Layer.RotateAngleToValue = "100 - (value / 360 * 100)"  
    var_Layer.ValueToRotateAngle = "(value)/100 * 360"  
    var_Layer.ValueToOffsetX = "value"  
    var_Layer.OffsetToValue = "value"  
    var_Layer.RotateAngleValid = "int(value / 360 * 100)/100 * 360"  
var_Layer1 = oGauge.Layers.Item(7)  
    var_Layer1.OnDrag = 2  
    var_Layer1.RotateType = 2  
oGauge.Value = 25  
oGauge.EndUpdate()
```

## Visual DataFlex

```
// Occurs when the layer's value is changed.
```

```
Procedure OnComChange Integer  IILayer
```

```
    Forward Send OnComChange IILayer
```

```
    Variant vA
```

```
        Get ComValue to vA
```

```
        Set ComExtraCaption "Client" OLEexLayerCaption to (ComFormatABC(Self,"<sha
```

;;0> <font ;12> <b>` + (100 - value format `0`)",vA,Nothing,Nothing))

Variant vA1

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComItem of hoLayers 9 to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Variant voLayer1

Get ComItem of hoLayer 9 to voLayer1

Handle hoLayer1

Get Create (RefClass(cComLayer)) to hoLayer1

Set pvComObject of hoLayer1 to voLayer1

Variant vX

Variant voLayers1

Get ComLayers to voLayers1

Handle hoLayers1

Get Create (RefClass(cComLayers)) to hoLayers1

Set pvComObject of hoLayers1 to voLayers1

Variant voLayer2

Get ComItem of hoLayers1 9 to voLayer2

Handle hoLayer2

Get Create (RefClass(cComLayer)) to hoLayer2

Set pvComObject of hoLayer2 to voLayer2

Variant voLayer3

Get ComItem of hoLayer2 9 to voLayer3

Handle hoLayer3

Get Create (RefClass(cComLayer)) to hoLayer3

Set pvComObject of hoLayer3 to voLayer3

Get ComRotamoveCenterX of hoLayer3 to vX

Send Destroy to hoLayer3

Send Destroy to hoLayer2

Send Destroy to hoLayers1

Variant vY

Variant voLayers2

Get ComLayers to voLayers2

Handle hoLayers2

Get Create (RefClass(cComLayers)) to hoLayers2

Set pvComObject of hoLayers2 to voLayers2

Variant voLayer4

Get ComItem of hoLayers2 9 to voLayer4

Handle hoLayer4

Get Create (RefClass(cComLayer)) to hoLayer4

Set pvComObject of hoLayer4 to voLayer4

Variant voLayer5

Get ComItem of hoLayer4 9 to voLayer5

Handle hoLayer5

Get Create (RefClass(cComLayer)) to hoLayer5

Set pvComObject of hoLayer5 to voLayer5

Get ComRotamoveCenterY of hoLayer5 to vY

Send Destroy to hoLayer5

Send Destroy to hoLayer4

Send Destroy to hoLayers2

Get ComLayerToClientX of hoLayer1 vX vY to vA1

Send Destroy to hoLayer1

Send Destroy to hoLayer

Send Destroy to hoLayers

Set ComExtraCaption "Client" OLEexLayerCaptionLeft to  
(ComFormatABC(Self,"value - 8",vA1,Nothing,Nothing))

Variant vA2

Variant voLayers3

Get ComLayers to voLayers3

Handle hoLayers3

Get Create (RefClass(cComLayers)) to hoLayers3

Set pvComObject of hoLayers3 to voLayers3

Variant voLayer6

Get ComItem of hoLayers3 9 to voLayer6

Handle hoLayer6

Get Create (RefClass(cComLayer)) to hoLayer6

Set pvComObject of hoLayer6 to voLayer6

Variant voLayer7

Get ComItem of hoLayer6 9 to voLayer7

Handle hoLayer7

Get Create (RefClass(cComLayer)) to hoLayer7

Set pvComObject of hoLayer7 to voLayer7

Variant vX1

Variant voLayers4

Get ComLayers to voLayers4

Handle hoLayers4

Get Create (RefClass(cComLayers)) to hoLayers4

Set pvComObject of hoLayers4 to voLayers4

Variant voLayer8

Get ComItem of hoLayers4 9 to voLayer8

Handle hoLayer8

Get Create (RefClass(cComLayer)) to hoLayer8

Set pvComObject of hoLayer8 to voLayer8

Variant voLayer9

Get ComItem of hoLayer8 9 to voLayer9

Handle hoLayer9

Get Create (RefClass(cComLayer)) to hoLayer9

Set pvComObject of hoLayer9 to voLayer9

Get ComRotamoveCenterX of hoLayer9 to vX1

Send Destroy to hoLayer9

Send Destroy to hoLayer8

Send Destroy to hoLayers4

Variant vY1

Variant voLayers5

Get ComLayers to voLayers5

Handle hoLayers5

Get Create (RefClass(cComLayers)) to hoLayers5

Set pvComObject of hoLayers5 to voLayers5

Variant voLayer10

Get ComItem of hoLayers5 9 to voLayer10

Handle hoLayer10

Get Create (RefClass(cComLayer)) to hoLayer10

Set pvComObject of hoLayer10 to voLayer10

Variant voLayer11

```

        Get ComItem of hoLayer10 9 to voLayer11
        Handle hoLayer11
        Get Create (RefClass(cComLayer)) to hoLayer11
        Set pvComObject of hoLayer11 to voLayer11
            Get ComRotamoveCenterY of hoLayer11 to vY1
        Send Destroy to hoLayer11
    Send Destroy to hoLayer10
    Send Destroy to hoLayers5
    Get ComLayerToClientY of hoLayer7 vX1 vY1 to vA2
    Send Destroy to hoLayer7
    Send Destroy to hoLayer6
    Send Destroy to hoLayers3
    Set ComExtraCaption "Client" OLEexLayerCaptionTop to
(ComFormatABC(Self,"value - 26",vA2,Nothing,Nothing))
End_Procedure

```

#### Procedure OnCreate

```

    Forward Send OnCreate
    Send ComBeginUpdate
    Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    Set ComPicturesName to "`Layer` + str(value + 1) + `.png`"
    Variant voLayers6
    Get ComLayers to voLayers6
    Handle hoLayers6
    Get Create (RefClass(cComLayers)) to hoLayers6
    Set pvComObject of hoLayers6 to voLayers6
        Set ComCount of hoLayers6 to 11
    Send Destroy to hoLayers6
    Set ComAllowSmoothChange to OLEexSmoothChangeless
    Variant voLayers7
    Get ComLayers to voLayers7
    Handle hoLayers7
    Get Create (RefClass(cComLayers)) to hoLayers7
    Set pvComObject of hoLayers7 to voLayers7
        Variant voLayer12
        Get ComItem of hoLayers7 9 to voLayer12

```



```

Handle hoLayer12
Get Create (RefClass(cComLayer)) to hoLayer12
Set pvComObject of hoLayer12 to voLayer12
    Set ComDefaultRotateAngle of hoLayer12 to -126
    Set ComOnDrag of hoLayer12 to OLEexDoRotamove
    Set ComRotateAngleToValue of hoLayer12 to "100 - (value / 360 * 100)"
    Set ComValueToRotateAngle of hoLayer12 to "(value)/100 * 360"
    Set ComValueToOffsetX of hoLayer12 to "value"
    Set ComOffsetToValue of hoLayer12 to "value"
    Set ComRotateAngleValid of hoLayer12 to "int(value / 360 * 100)/100 * 360"
Send Destroy to hoLayer12
Send Destroy to hoLayers7
Variant voLayers8
Get ComLayers to voLayers8
Handle hoLayers8
Get Create (RefClass(cComLayers)) to hoLayers8
Set pvComObject of hoLayers8 to voLayers8
    Variant voLayer13
    Get ComItem of hoLayers8 7 to voLayer13
    Handle hoLayer13
    Get Create (RefClass(cComLayer)) to hoLayer13
    Set pvComObject of hoLayer13 to voLayer13
        Set ComOnDrag of hoLayer13 to OLEexDoRotate
        Set ComRotateType of hoLayer13 to OLEexRotateBilinearInterpolation
    Send Destroy to hoLayer13
Send Destroy to hoLayers8
Set ComValue to 25
Send ComEndUpdate
End_Procedure

```

## XBase++

```

PROCEDURE OnChange(oGauge,Layer)

```

```

oGauge:SetProperty("ExtraCaption","Client",0/*exLayerCaption*/,oGauge:FormatABC("
;;0> <font ;12> <b>` + (100 - value format `0`)",oGauge:Value()))

```

```

oGauge:SetProperty("ExtraCaption","Client",4/*exLayerCaptionLeft*/,oGauge:FormatA
-
8",oGauge:Layers:Item(9):LayerToClientX(oGauge:Layers:Item(9):RotamoveCenterX(),oG
oGauge:SetProperty("ExtraCaption","Client",5/*exLayerCaptionTop*/,oGauge:FormatA
-
26",oGauge:Layers:Item(9):LayerToClientY(oGauge:Layers:Item(9):RotamoveCenterX(),oG

```

```

RETURN

```

```

#include "AppEvent.ch"
#include "ActiveX.ch"

```

```

PROCEDURE Main

```

```

    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge
    LOCAL oLayer,oLayer1

```

```

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

```

```

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

```

```

        oGauge:Change := {|| Layer| OnChange(oGauge,Layer)} /*Occurs when the layer's
value is changed.*/

```

```

        oGauge:BeginUpdate()
        oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        oGauge:PicturesName := ""Layer` + str(value + 1) + `.png""

```

```
oGauge:Layers():Count := 11
oGauge:AllowSmoothChange := 0/*exSmoothChangeless*/
oLayer := oGauge:Layers:Item(9)
  oLayer:DefaultRotateAngle := -126
  oLayer:OnDrag := 3/*exDoRotamove*/
  oLayer:RotateAngleToValue := "100 - (value / 360 * 100)"
  oLayer:ValueToRotateAngle := "(value)/100 * 360"
  oLayer:ValueToOffsetX := "value"
  oLayer:OffsetToValue := "value"
  oLayer:RotateAngleValid := "int(value / 360 * 100)/100 * 360"
oLayer1 := oGauge:Layers:Item(7)
  oLayer1:OnDrag := 2/*exDoRotate*/
  oLayer1:RotateType := 2/*exRotateBilinearInterpolation*/
oGauge:Value := 25
oGauge:EndUpdate()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

## property Layer.Left as String

Specifies the expression relative to the view, to determine the x-position to show the current layer on the control.

Type	Description
String	A String expression expression relative to the view, to determine the x-position to show the current layer on the control.

By default, the Left property is "0". If the Left property is empty, missing or invalid, it is considered "0". If valid, the value of evaluating the Left property indicates the left position of the layer as shown in the picture bellow. Use the [DefaultLayer\(exDefLayerLeft\)](#) property to specify the default value for the Left property, before adding any layer.

For instance:

- "0" indicates the left side of the control's view
- "width / 2", half of the view or center of the control's view
- "width - 64", 64 pixels to the right side of the control's view

The Left property supports the following keywords:

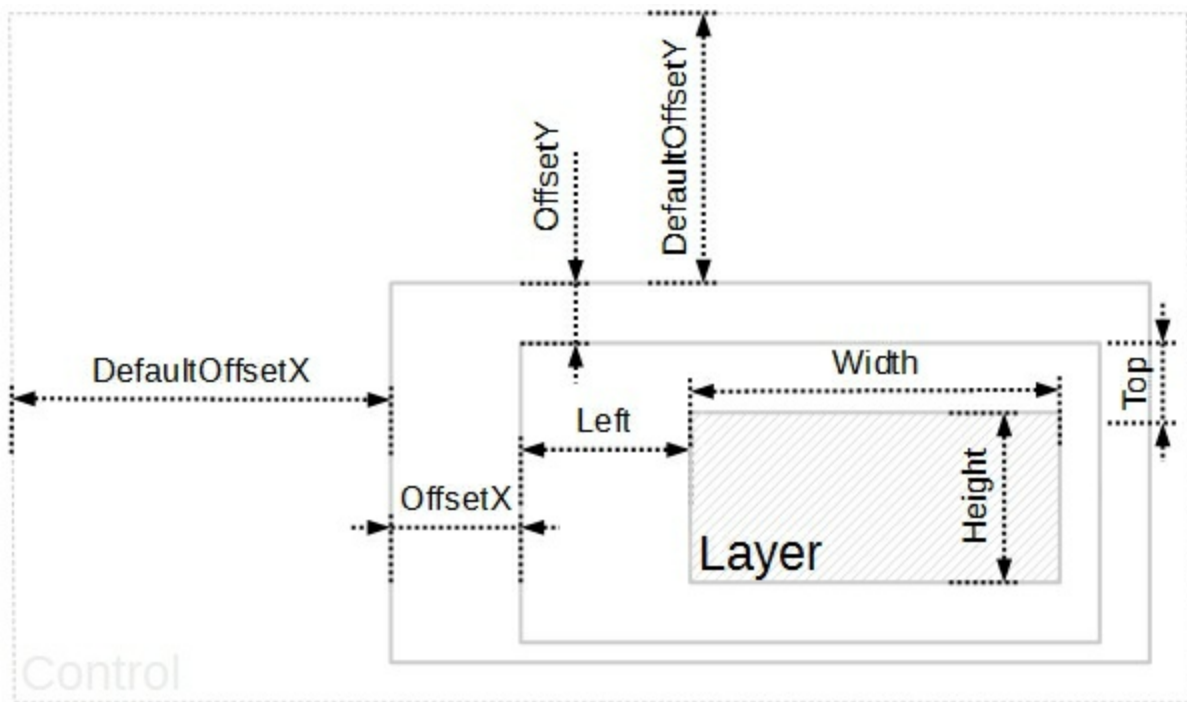
- **width** keyword specifies the width in pixels of the control's view
- **height** keyword specifies the height in pixels of the control's view

Also, this property supports all constants, operators and functions defined [here](#).

The following properties determines the position / size / offset of the **layer**:

- Left, specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

The following picture shows the position/size properties of the Layer, relative to the view / control:



You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

# property Layer.OffsetToValue as String

Specifies the expression to convert the offsetx,offsety to value.

Type	Description
String	A String value that specifies the expression to convert the offsetx,offsety to value.

By default, the OffsetToValue property is empty. If the OffsetToValue property is empty, missing or invalid, it has no effect. If the OffsetToValue property is valid, the result of evaluation of the OffsetToValue property indicates the layer's [Value](#) property. The [ValueToOffsetX](#) / [ValueToOffsetY](#) property converts the value back to an offset. Use the [DefaultLayer\(exDefLayerOffsetToValue\)](#) property to specify the default value for the OffsetToValue property, before adding the layer.

For instance:

- "0", specifies that the layer's value is always 0 no matter of the offset of the layer
- "value = 0 ? 0 : 1", equivalent with "offsetx = 0 ? 0 : 1" indicates that if the layer's OffsetX is 0, the value is 0 else it is one
- "offsety = 0 ? 0 : 1", indicates that if the layer's OffsetY is 0, the value is 0 else it is one

The OffsetToValue property supports the following keywords:

- **value** or **offsetx** keyword indicates the layer's [OffsetX](#) property
- **offsety** keyword indicates the layer's [OffsetY](#) property

Also, this property supports all constants, operators and functions defined [here](#).

The [Value](#) property indicates the **value** keyword in the following properties:

- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the [ValueToOffsetX](#) expression, while the [OnDrag](#) property is **exDoMove**. The OffsetToValue converts the current offset to a value.
- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is exDoMove. The OffsetToValue converts the current offset to a value.
- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is exDoRotate or exDoRotamove. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The Value property works in association with the layer's [OnDrag](#) property like follows:

- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetX](#) property indicates the layer's [OffsetX](#) property.
- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetY](#) property indicates the layer's [OffsetY](#) property.
- If the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**, evaluating the [ValueToRotateAngle](#) property indicates the layer's [RotateAngle](#) property.

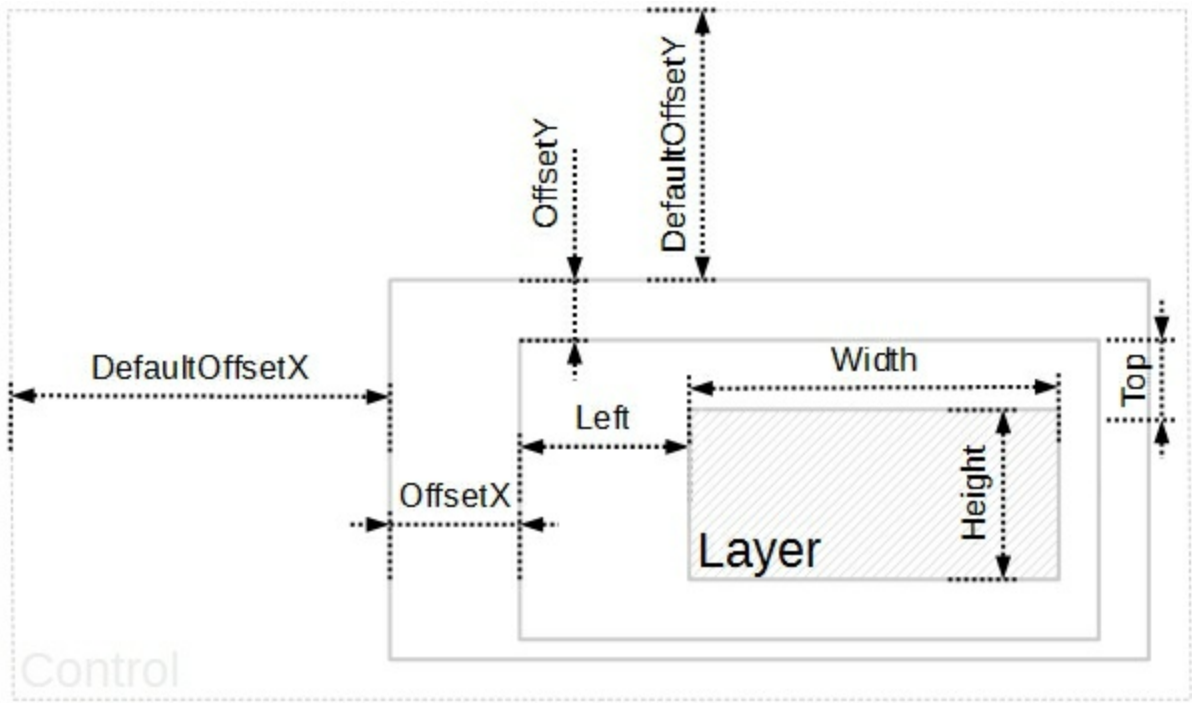
# property Layer.OffsetX as Long

Gets or sets a value that indicates x-offset of the layer.

Type	Description
Long	A Long expression that indicates x-offset of the layer.

By default, the OffsetX / OffsetY property is 0. The OffsetX / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. The [Value](#) property associates a value to a layer. The [ValueToOffsetX](#) property specifies the expression to convert the value to x-offset. The [ValueToOffsetY](#) property specifies the expression to convert the value to y-offset. For instance, you can use the OffsetYValid property on "0", and so no vertical movement is allowed for the current layer. Use the [DefaultLayer\(exDefLayerOffsetX\)](#) property to specify the default value for the OffsetX, before adding any layer. The [Change](#) event occurs when the layer's OffsetX property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. The layer's [RotamoveOffsetX](#) / [RotamoveOffsetY](#) property indicates the current (x,y) position of the layer, while the [OnDrag](#) property is exDoRotamove.

The following picture shows the position/size properties of the Layer, relative to the view / control:



The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show



the current layer on the control.

- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

# property Layer.OffsetXValid as String

Validates the x-offset value of the layer

Type	Description
String	A String expression that validates the x-offset value of the layer. The result of evaluating the expression indicates the newly <a href="#">OffsetX</a> value.

By default, the [OffsetXValid](#) / [OffsetYValid](#) property is empty. The [OffsetXValid](#) / [OffsetYValid](#) property has no effect if it is empty, missing or invalid. If the [OffsetXValid](#) / [OffsetYValid](#) property is valid expression, the value of [OffsetX](#) property always matches [OffsetXValid](#) expression. In other words, the [OffsetXValid](#) validates the x-position of the layer. For instance, you can use the [OffsetXValid](#) / [OffsetYValid](#) expression to specify a range of values to allow the [OffsetX](#) / [OffsetY](#) properties. Use the [DefaultLayer\(exDefLayerOffsetXValid\)](#) property to specify the default value for the [OffsetXValid](#) property, before adding any layer.

The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [Value](#) property associates a value to a layer. The [ValueToOffsetX](#) property specifies the expression to convert the value to x-offset. The [ValueToOffsetY](#) property specifies the expression to convert the value to y-offset. For instance, you can use the [OffsetYValid](#) property on "0", and so no vertical movement is allowed for the current layer. Use the [DefaultLayer\(exDefLayerOffsetX\)](#) property to specify the default value for the [OffsetX](#), before adding any layer. The [Change](#) event occurs when the layer's [OffsetX](#) property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged.

For instance:

- "0", indicates that no horizontal movement is allowed, or in other words, [OffsetX](#) is always 0.
- "16 \* int(value / 16)", specifies that only multiply of 16 is allowed for [OffsetX](#) property ( grid movement )
- "value = 0 ? 0 : ( value < 0 ? -100 : +100 )", indicates that valid values for [OffsetX](#) property is -100, 0 and +100 ( discrete movement )
- "value MIN -64 MAX 64", indicates that values of [OffsetX](#) property are between -64 and +64 ( range movement )
- "y" indicates that [OffsetX](#) property is always the same as [OffsetY](#) property ( diagonal movement )

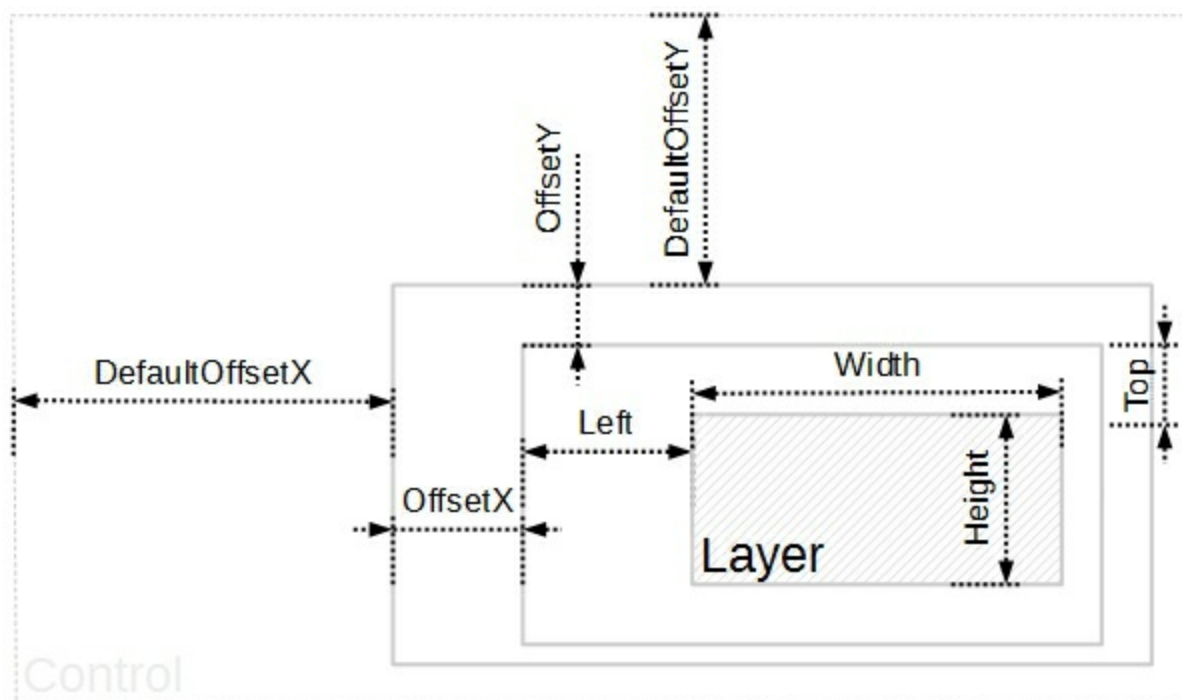
The [OffsetXValid](#) property supports the following keywords:

- **value** keyword indicates the current value of [OffsetX](#) property ( the value to validate )

- **y** keyword specifies the current value of the [OffsetY](#) property

Also, this property supports all constants, operators and functions defined [here](#).

The following picture shows the position/size properties of the Layer, relative to the view / control:



The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.

- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

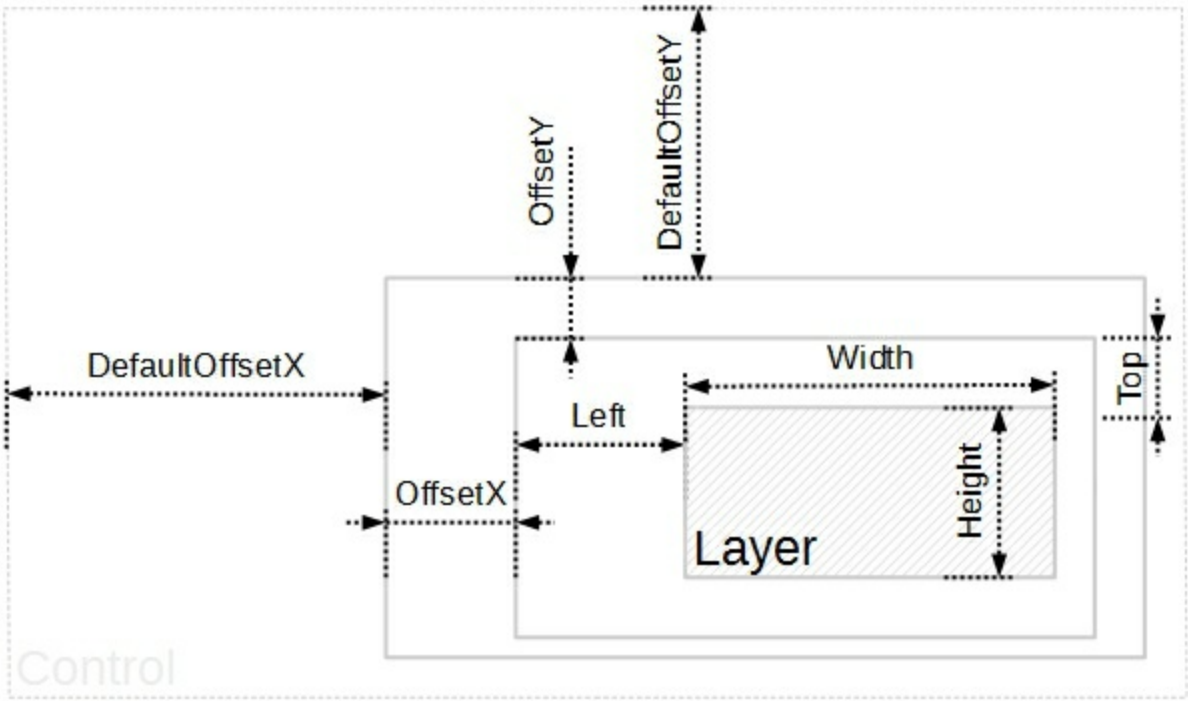
# property Layer.OffsetY as Long

Gets or sets a value that indicates y-offset of the layer.

Type	Description
Long	A Long expression that indicates y-offset of the layer.

By default, the OffsetX / OffsetY property is 0. The [OffsetX](#) / OffsetY property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [OffsetXValid](#) / [OffsetYValid](#) property to validate the (x,y)-position of the layer. The [Value](#) property associates a value to a layer. The [ValueToOffsetX](#) property specifies the expression to convert the value to x-offset. The [ValueToOffsetY](#) property specifies the expression to convert the value to y-offset. For instance, you can use the OffsetXValid property on "0", and so no horizontal movement is allowed for the current layer. Use the [DefaultLayer\(exDefLayerOffsetY\)](#) property to specify the default value for the OffsetY, before adding any layer. The [Change](#) event occurs when the layer's OffsetY property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. The layer's [RotamoveOffsetX](#) / [RotamoveOffsetY](#) property indicates the current (x,y) position of the layer, while the [OnDrag](#) property is exDoRotamove.

The following picture shows the position/size properties of the Layer, relative to the view / control:



The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show

the current layer on the control.

- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

# property Layer.OffsetYValid as String

Validates the y-offset value of the layer

Type	Description
String	A String expression that validates the y-offset value of the layer. The result of evaluating the expression indicates the newly <a href="#">OffsetY</a> value.

By default, the [OffsetXValid](#) / [OffsetYValid](#) property is empty. The [OffsetXValid](#) / [OffsetYValid](#) property has no effect if it is empty, missing or invalid. If the [OffsetXValid](#) / [OffsetYValid](#) property is valid expression, the value of [OffsetY](#) property always matches [OffsetYValid](#) expression. In other words, the [OffsetYValid](#) validates the y-position of the layer. For instance, you can use the [OffsetXValid](#) / [OffsetYValid](#) expression to specify a range of values to allow the [OffsetX](#) / [OffsetY](#) properties. Use the [DefaultLayer\(exDefLayerOffsetYValid\)](#) property to specify the default value for the [OffsetYValid](#) property, before adding any layer.

The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [Value](#) property associates a value to a layer. The [ValueToOffsetX](#) property specifies the expression to convert the value to x-offset. The [ValueToOffsetY](#) property specifies the expression to convert the value to y-offset. For instance, you can use the [OffsetYValid](#) property on "0", and so no vertical movement is allowed for the current layer. Use the [DefaultLayer\(exDefLayerOffsetX\)](#) property to specify the default value for the [OffsetX](#), before adding any layer. The [Change](#) event occurs when the layer's [OffsetX](#) property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged.

For instance:

- "0", indicates that no vertical movement is allowed, or in other words, [OffsetY](#) is always 0.
- "16 \* int(value / 16)", specifies that only multiply of 16 is allowed for [OffsetY](#) property ( grid movement )
- "value = 0 ? 0 : ( value < 0 ? -100 : +100 )", indicates that valid values for [OffsetY](#) property is -100, 0 and +100 ( discrete movement )
- "value MIN -64 MAX 64", indicates that values of [OffsetY](#) property are between -64 and +64 ( range movement )
- "x" indicates that [OffsetY](#) property is always the same as [OffsetX](#) property ( diagonal movement )

The [OffsetYValid](#) property supports the following keywords:

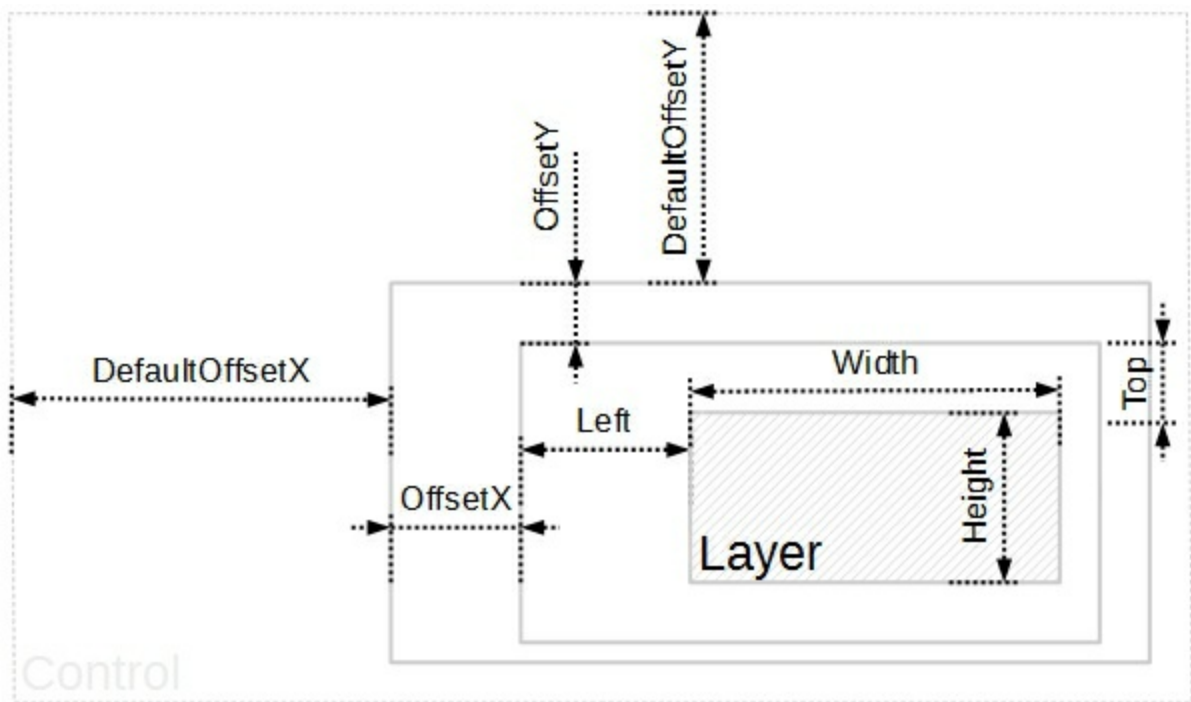
- **value** keyword indicates the current value of [OffsetY](#) property ( the value to validate )



- **x** keyword specifies the current value of the [OffsetX](#) property

Also, this property supports all constants, operators and functions defined [here](#).

The following picture shows the position/size properties of the Layer, relative to the view / control:



The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.



- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

# property Layer.OnDrag as OnDragLayerEnum

Indicates the action to be performed when the user drags the layer.

Type	Description
<a href="#">OnDragLayerEnum</a>	An OnDragLayerEnum expression that specifies the operation to perform when the user clicks and drags the layer.

By default, the OnDrag property is `exDoNothing`, so nothing is happen if the user clicks the layer. The [Change](#) event occurs when the layer's value property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. The [ShowHandCursor](#) property returns or sets a value that indicates whether the hand cursor is shown when it hovers a visible / selectable / draggable layer. The [Clip](#) property gets access to the layer's clipping object. The [Value](#) property of the Layer has effect only if the OnDrag property is not `exDoNothing`. The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable.

Currently, any layer supports any of the following operations:

- **exDoMove**, moves a layer, the layer's [OffsetX](#) and [OffsetY](#) indicates the current (x,y) position of the layer.
- **exDoRotate**, rotates a layer, the [RotateAngle](#) property indicates the currently rotation angle.
- **exDoRotamove**, rotates the layer by moving, the [RotateAngle](#) property indicates the currently rotation angle. In this case, the layer's [RotamoveOffsetX](#) / [RotamoveOffsetY](#) property indicates the current (x,y) position of the layer. The `exDoRotamove` operation does not actually rotate the layer's view instead it moves / rotates it relative to its center.

The control fires the drag events in the following order:

- [DragStart](#) event notifies that a layer begins to drag. You can use the DragStart event to cancel the dragging operation.
- [Drag](#) event notifies that the layer is dragging. You can use the Drag event to perform other actions, on any layer during the dragging operation.
- [DragEnd](#) event notifies that the dragging the layer ends. You can use the DragEnd event to perform other actions, on any layer when dragging operation ends.

You can use the [Debug](#) property of the DragInfo object to display debugging information during dragging.

# property Layer.Position as Long

Retrieves or sets a value that indicates the position/z-order of the layer in the control.

Type	Description
Long	A Long expression that specifies the position of the layer within the layers collection.

The Position property specifies the position of the layer, in the layers collection. The [Visible](#) property shows or hides a specific layer (visible). The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [LayerFromPoint](#) property retrieves the layer from point that's visible and selectable. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( dragable ).

# property Layer.RotamoveCenterX as Long

Specifies the x-position of the layer's center, while the layer's drag operation is exDoRotamove.

Type	Description
Long	A Long expression that specifies the x-position of the layer's center

By default the RotamoveCenterX / [RotamoveCenterY](#) indicates the (x,y)-center of the layer's view. The control supports moving the layer by rotation, also called rotamove. The rotamove operation moves layer and the rotamove center pointed by RotamoveCenterX / RotamoveCenterY properties around the rotation center pointed by the [RotateCenterX](#) / [RotateCenterY](#) properties of the [RotateCenterLayer](#) layer. The [LayerToClientX](#) / [LayerToClientY](#) properties translate a point from the layer ( as it is moved or rotated ) to the control's view.

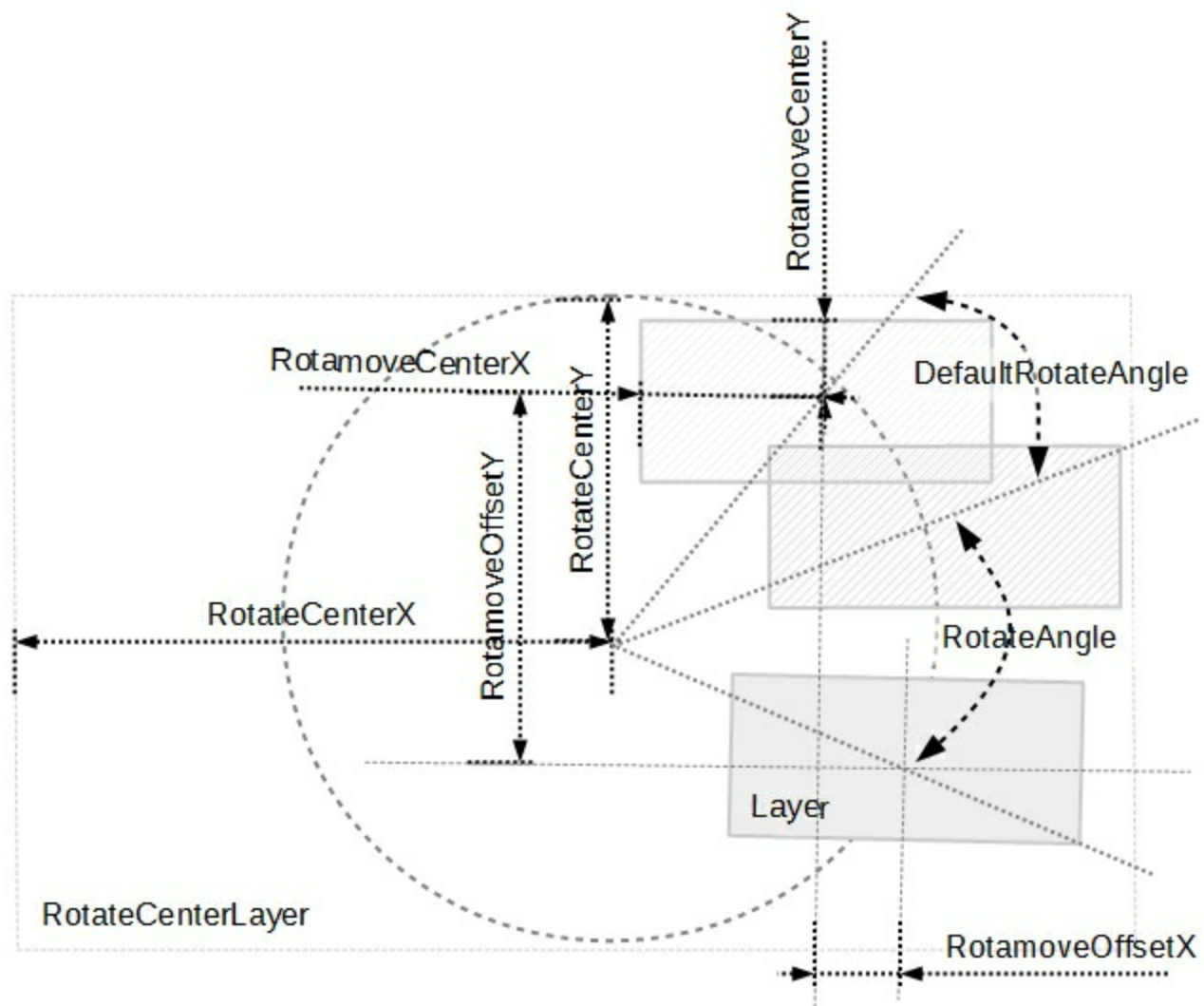
The center of the layer may be different than layer's view, as for instance, if you have a layer that's shows a knob in the bottom-side of the layer the RotamoveCenterX / RotamoveCenterY point is the center on the knob in the bottom-right side not in the center of the layer as you can see in the following screen shot ( red cross ) :



The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- RotamoveCenterX, specifies the x-position of the layer's center ( view ).
- [RotamoveCenterY](#), specifies the y-position of the layer's center ( view ).
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

The following picture shows the rotamove properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

# property Layer.RotamoveCenterY as Long

Specifies the y-position of the layer's center, while the layer's drag operation is exDoRotamove.

Type	Description
Long	A Long expression that specifies the x-position of the layer's center

By default the [RotamoveCenterX](#) / RotamoveCenterY indicates the (x,y)-center of the layer's view. The control supports moving the layer by rotation, also called rotamove. The rotamove operation moves layer and the rotamove center pointed by RotamoveCenterX / RotamoveCenterY properties around the rotation center pointed by the [RotateCenterX](#) / [RotateCenterY](#) properties of the [RotateCenterLayer](#) layer. The [LayerToClientX](#) / [LayerToClientY](#) properties translate a point from the layer ( as it is moved or rotated ) to the control's view.

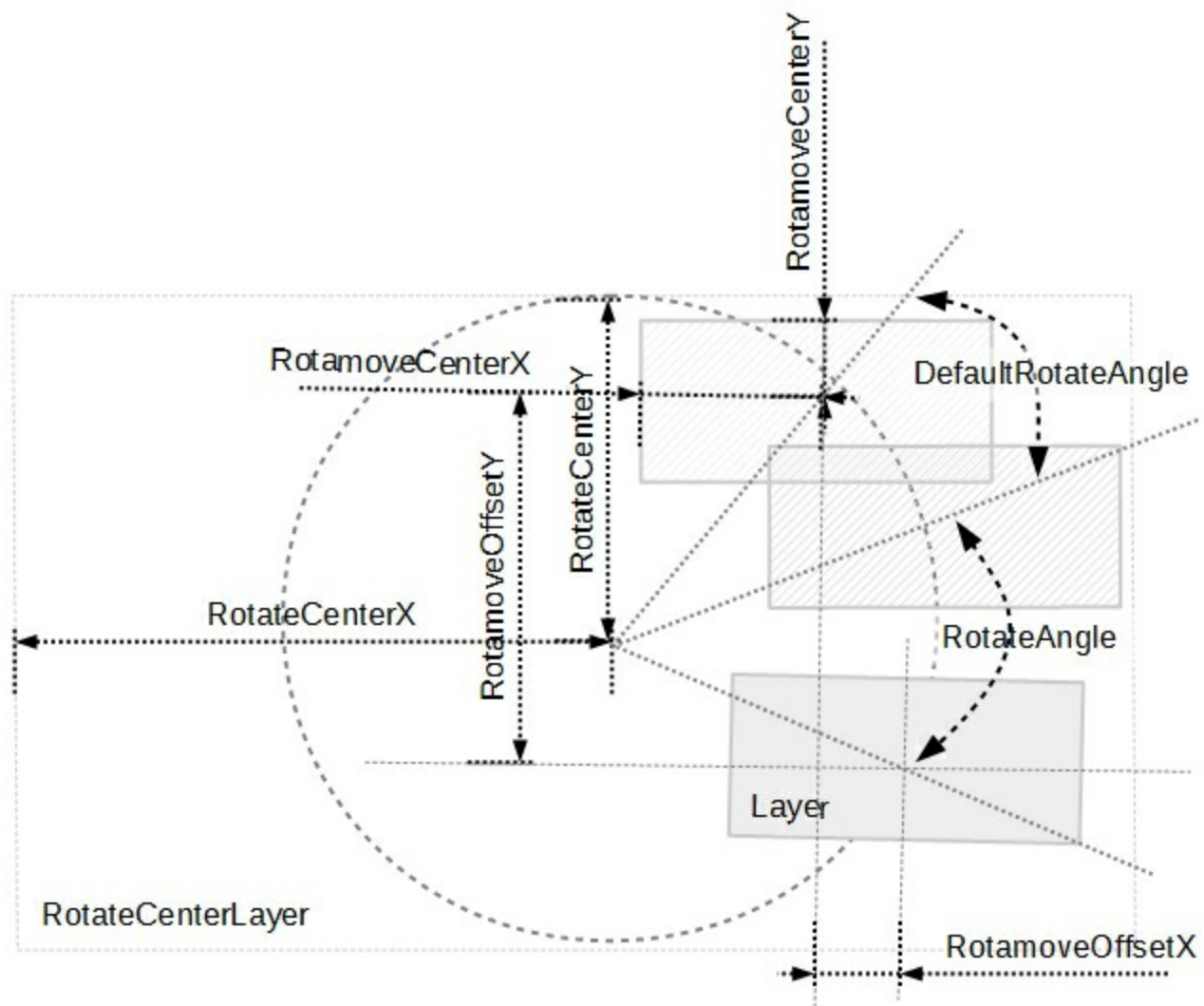
The center of the layer may be different than layer's view, as for instance, if you have a layer that's shows a knob in the bottom-side of the layer the RotamoveCenterX / RotamoveCenterY point is the center on the knob in the bottom-right side not in the center of the layer as you can see in the following screen shot ( red cross ) :



The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center ( view ).
- RotamoveCenterY, specifies the y-position of the layer's center ( view ).
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

The following picture shows the rotamove properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

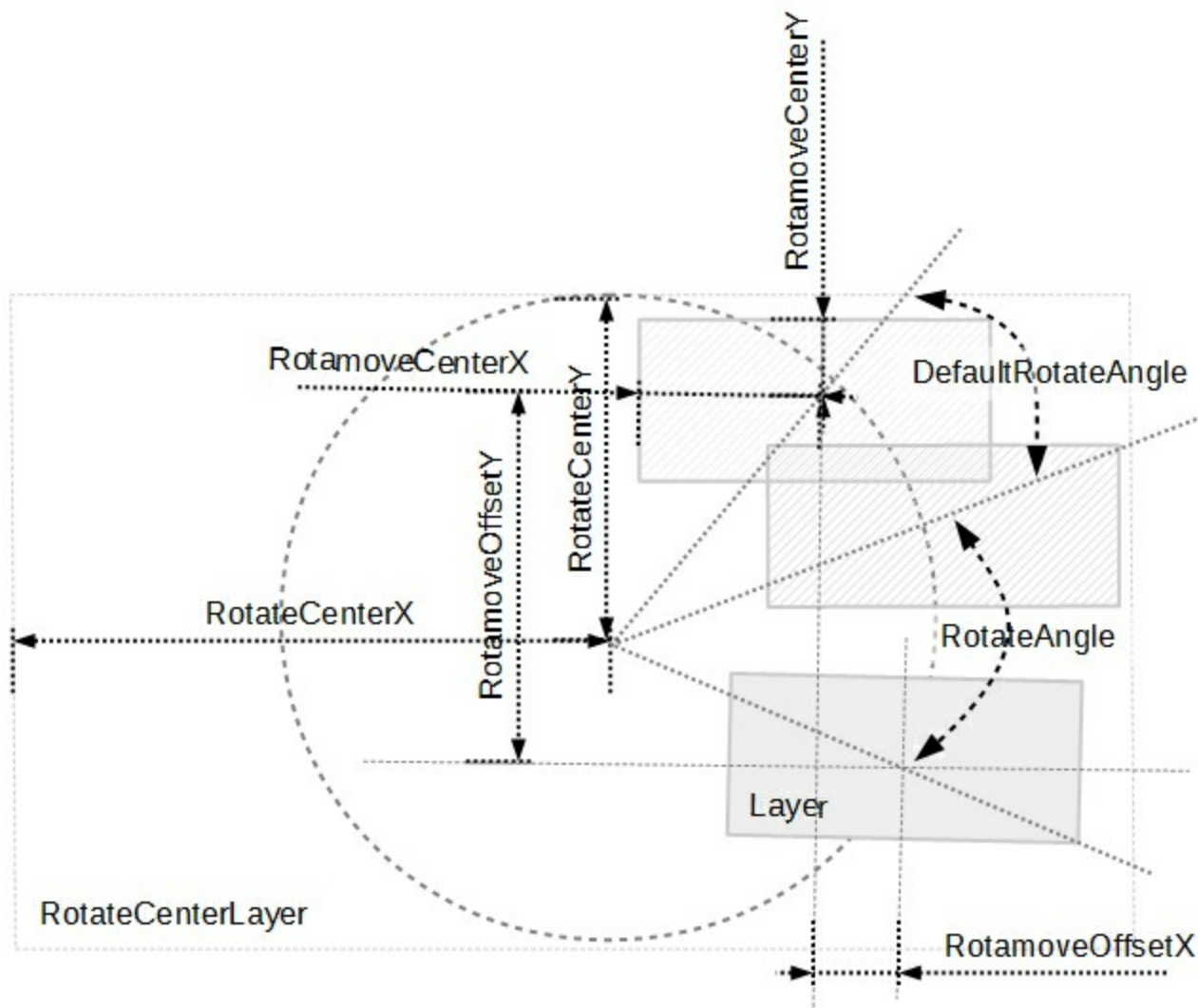


# property Layer.RotamoveOffsetX as Long

Specifies the x-offset of the layer, while the layer's drag operation is exDoRotamove.

Type	Description
Long	A Long expression that specifies the x-offset of the layer, while the layer's drag operation is exDoRotamove.

The control supports moving the layer by rotation, also called rotamove. The rotamove operation moves layer and the rotamove center pointed by [RotamoveCenterX](#) / [RotamoveCenterY](#) properties around the rotation center pointed by the [RotateCenterX](#) / [RotateCenterY](#) properties of the [RotateCenterLayer](#) layer. The following picture shows the rotamove properties of the Layer, relative to the [RotateCenterLayer](#) layer:



The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center ( view ).
- [RotamoveCenterY](#), specifies the y-position of the layer's center ( view ).
- [RotamoveOffsetX](#), specifies the x-offset of the layer.



- [RotamoveOffsetY](#), specifies the y-offset of the layer.

Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

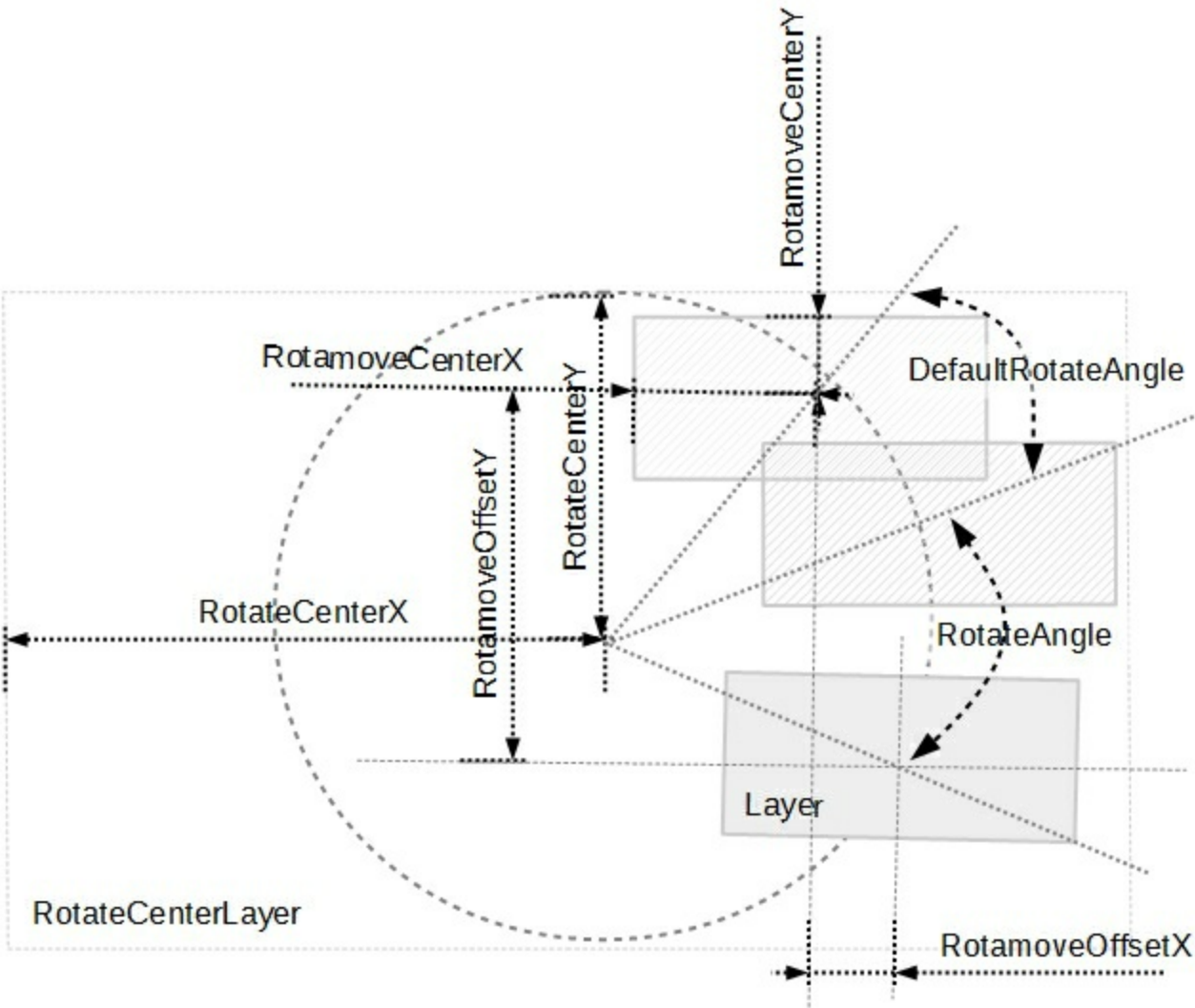
- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

# property Layer.RotamoveOffsetY as Long

Specifies the y-offset of the layer, while the layer's drag operation is exDoRotamove.

Type	Description
Long	A Long expression that specifies the y-offset of the layer, while the layer's drag operation is exDoRotamove.

The control supports moving the layer by rotation, also called rotamove. The rotamove operation moves layer and the rotamove center pointed by [RotamoveCenterX](#) / [RotamoveCenterY](#) properties around the rotation center pointed by the [RotateCenterX](#) / [RotateCenterY](#) properties of the [RotateCenterLayer](#) layer. The following picture shows the rotamove properties of the Layer, relative to the [RotateCenterLayer](#) layer:



The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is `exDoRotamove`:

- [RotamoveCenterX](#), specifies the x-position of the layer's center ( view ).
- [RotamoveCenterY](#), specifies the y-position of the layer's center ( view ).
- [RotamoveOffsetX](#), specifies the x-offset of the layer.

- [RotamoveOffsetY](#), specifies the y-offset of the layer.

Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

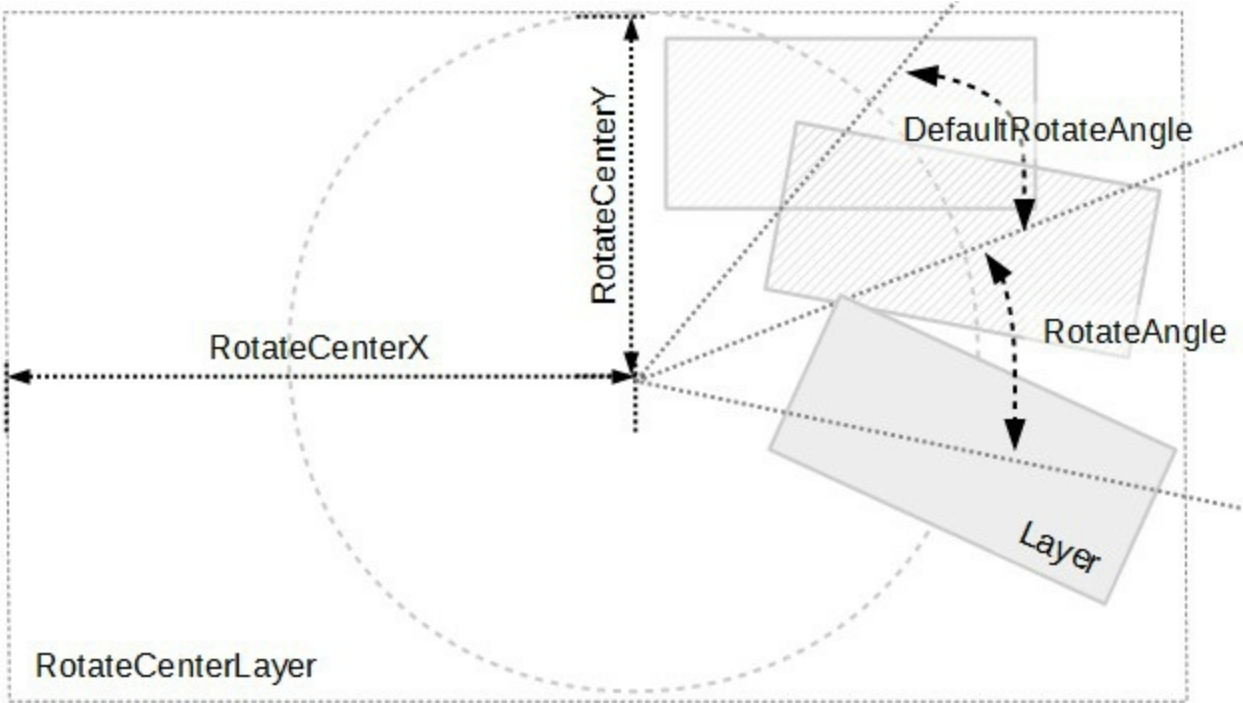
# property Layer.RotateAngle as Double

Specifies the angle to rotate the layer.

Type	Description
Double	A Double expression that specifies the angle to rotate the layer, in degree.

By default, the RotateAngle property is 0 degree ( which indicates that the layer is shown as it is ). The RotateAngle property specifies the current angle of the rotation of the specified layer. The [DeltaAngle](#) property specifies the angle (in degrees) that has been rotated the layer/object, during the drag operation. The [CumulativeRotateAngle](#) property specifies the cumulative rotation angle, during the dragging operation. The [Change](#) event occurs when the layer's RotateAngle property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. Change the [Debug](#) property of the [DragInfo](#) during the [DragStart](#) event to debug the rotation angles. Use the [DefaultLayer\(exDefLayerRotateAngle\)](#) property to specify the default value for the RotateAngle property, before adding any layer. The [RotateType](#) property specifies whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ...

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.

- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

# property Layer.RotateAngleToValue as String

Specifies the expression to convert the rotating angle to value.

Type	Description
String	A String value that specifies the expression to convert the rotating angle to value.

By default, the RotateAngleToValue property is empty. If the RotateAngleToValue property is empty, missing or invalid, it has no effect. If the RotateAngleToValue property is valid, the result of evaluation of the RotateAngleToValue property indicates the layer's [Value](#) property. The [ValueToRotateAngle](#) property converts the value back to a rotation angle. Use the [DefaultLayer\(exDefLayerRotateAngleToValue\)](#) property to specify the default value for the RotateAngleToValue property, before adding the layer.

For instance:

- "0", specifies that the layer's value is always 0 no matter of the rotation angle of the layer.
- "value / 360 \* 100", converts the angle to a percent
- "value / 360 / 24 / 60", converts the angle to a time value.

The RotateAngleToValue property supports the following keywords:

- **value** keyword indicates the layer's [RotateAngle](#) property

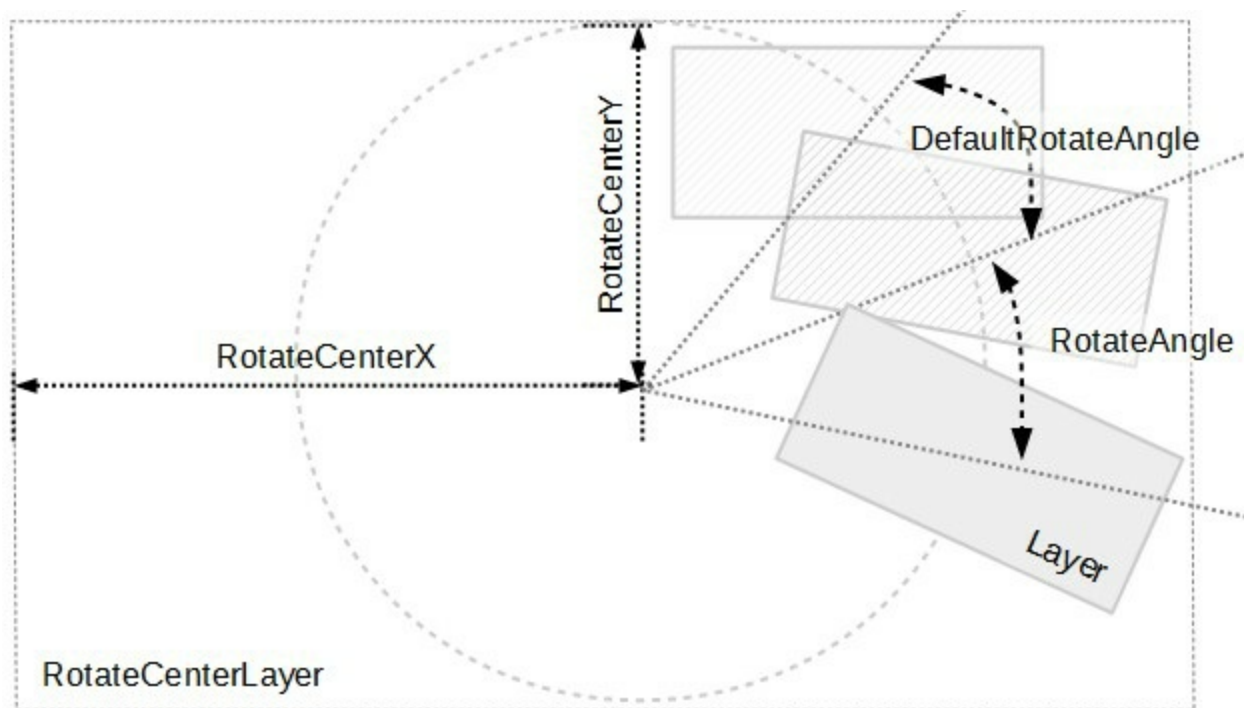
Also, this property supports all constants, operators and functions defined [here](#).

The [Value](#) property indicates the **value** keyword in the following properties:

- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is exDoRotate or exDoRotamove. The RotateAngleToValue converts the current rotation angle to a value.
- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the [ValueToOffsetX](#) expression, while the [OnDrag](#) property is **exDoMove**. The OffsetToValue converts the current offset to a value.
- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is exDoMove. The OffsetToValue converts the current offset to a value.

The following picture shows the rotation properties of the Layer, relative to the

[RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is `exDoRotamove`:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.



# property Layer.RotateAngleValid as String

Validates the rotation angle of the layer.

Type	Description
String	A String expression that validates the rotation angle value of the layer. The result of evaluating the expression indicates the newly <a href="#">RotateAngle</a> value.

By default, the RotateAngleValid property is empty. The RotateAngleValid property has no effect if it is empty, missing or invalid. If the RotateAngleValid property is valid expression, the value of [RotateAngle](#) property always matches RotateAngleValid expression. In other words, the RotateAngleValid validates the rotation angle of the layer. Use the [DefaultLayer\(exDefLayerRotateAngleValid\)](#) property to specify the default value for the RotateAngleValid property, before adding any layer. The [Change](#) event occurs when the layer's OffsetX property is changed. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. You can debug the rotation angle, using the [Debug](#) property of the [DragInfo](#) object. During drag operation you can use the [RotateAngleValid](#) property to limit the rotation angle.

For instance:

- "0", indicates that no rotation is allowed, or in other words, [RotateAngle](#) is always 0.
- "15 \* int(value / 15)", specifies that only multiply of 15 degree is allowed for [RotateAngle](#) property ( sectorial rotation )
- "value = 0 ? 0 : ( value < 0 ? -180 : +180 )", indicates that valid values for [RotateAngle](#) property is -180, 0 and +180 ( discrete rotation )
- "value MIN -90 MAX 90", indicates that values of [RotateAngle](#) property are between -90 and +90 degrees ( range movement )
- "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 : value ))", converts the value to an angle between 270 and 90 degree.
- "int(value / 360 \* 100)/100 \* 360" converts the value to an integer rotation angle
- "(value)/100 \* 360" converts a percent value to an angle

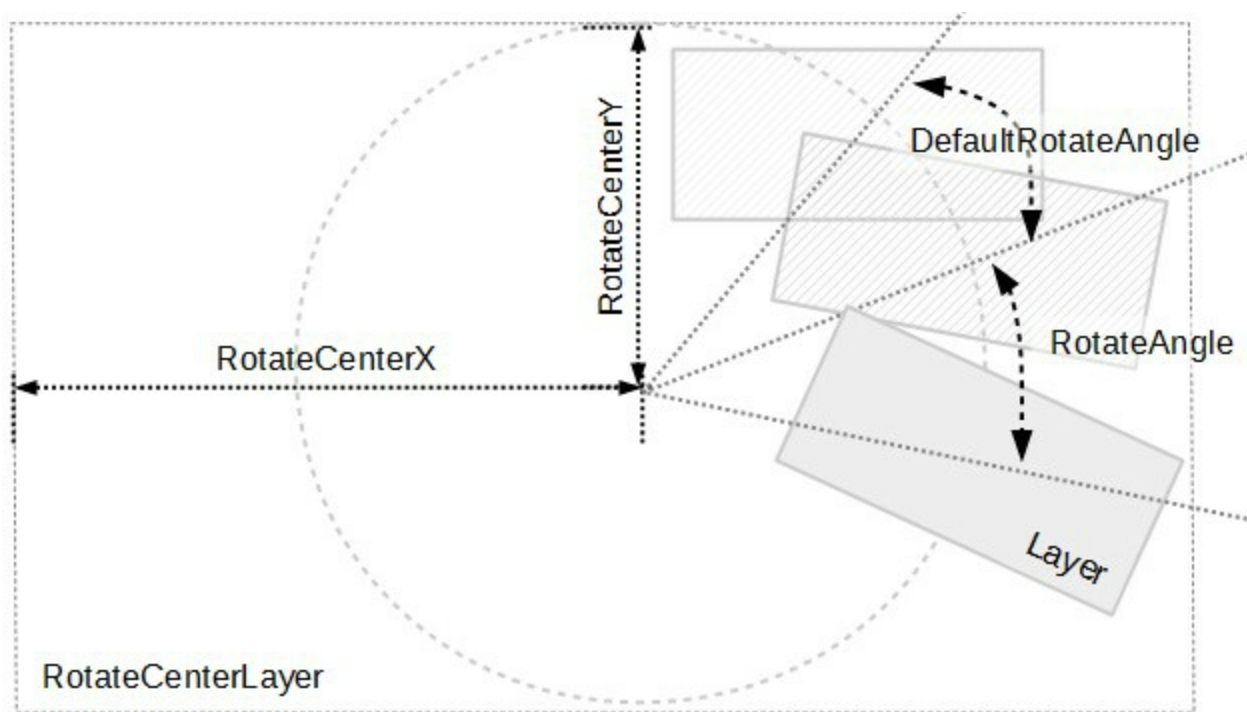
The RotateAngleValid property supports the following keywords:

- **value** keyword indicates the current value of [RotateAngle](#) property ( the value to validate )

Also, this property supports all constants, operators and functions defined [here](#).

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:





Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is `exDoRotamove`:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

# property Layer.RotateCenterLayer as Long

Indicates the index of the layer the rotation is around. If -1, the rotation is relative to the current layer.

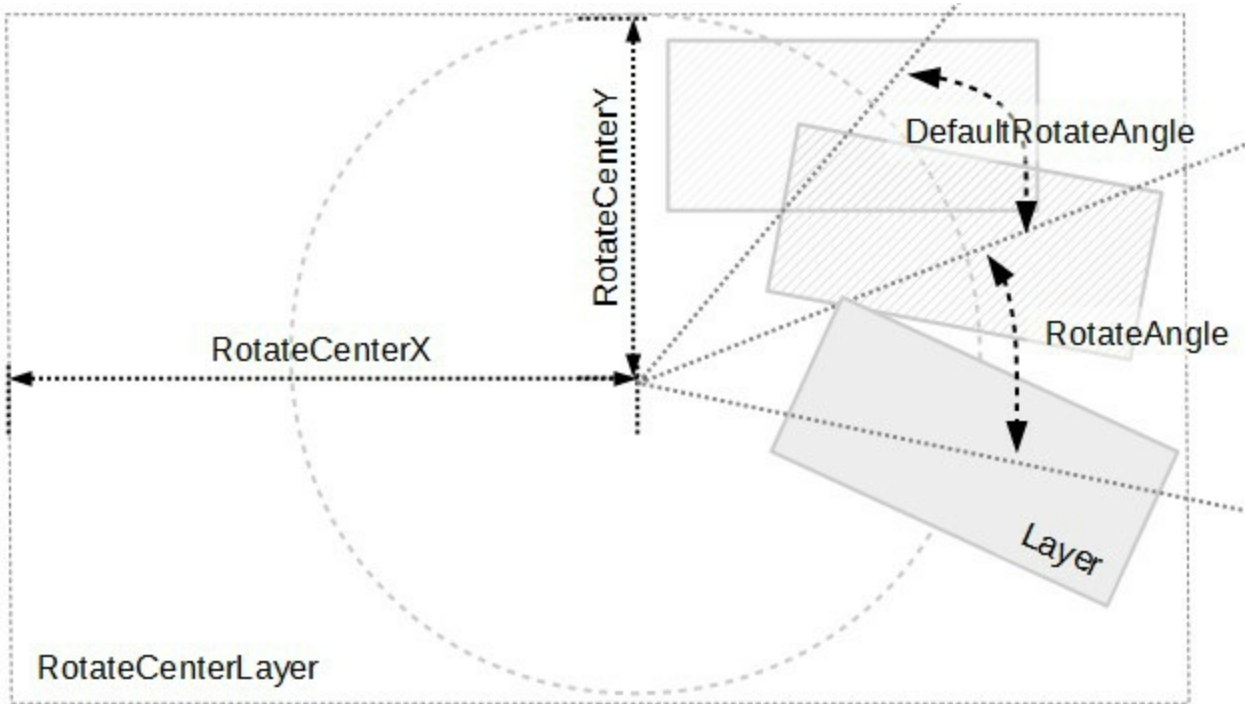
Type	Description
Long	A Long expression that indicates the index of the layer that holds the rotation center.

By default, the RotateCenterLayer property is 0, which indicates that all layers are rotated relative to the first layer. If the RotateCenterLayer property is -1, the rotation is performed around the layer itself. Use the [DefaultLayer\(exDefLayerRotateCenterLayer\)](#) property to specify the default value for the RotateCenterLayer property, before adding the layer.

The following properties can be used to specify a different rotation center:

- RotateCenterLayer, indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the RotateCenterLayer layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the RotateCenterLayer layer.

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.

- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and ValueToRotateAngle, specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

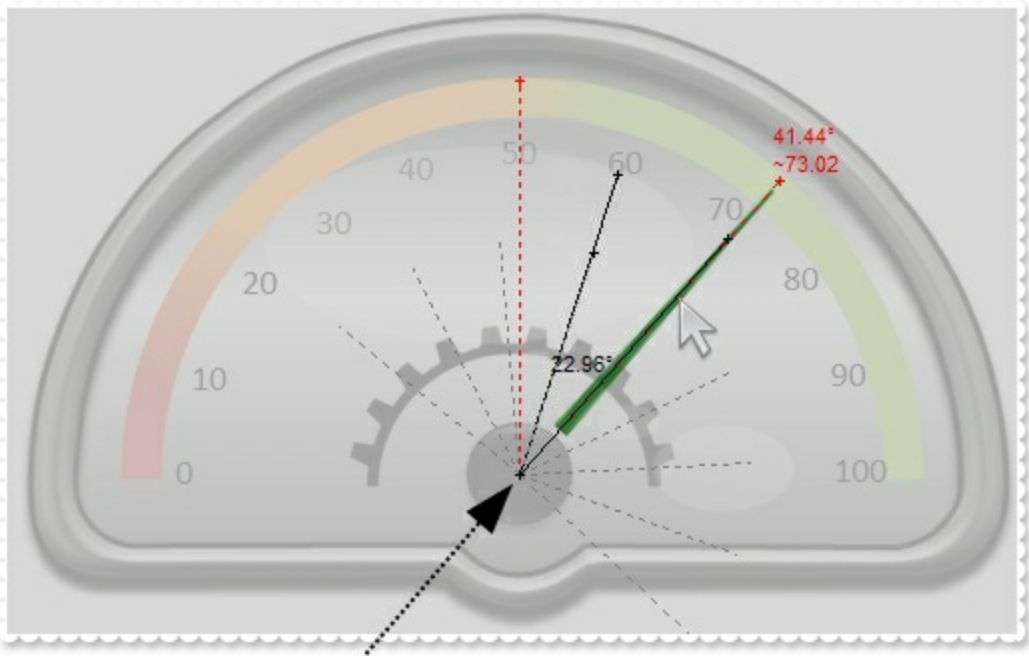
# property Layer.RotateCenterX as String

Indicates the expression that determines the x-origin of the rotation point relative to the RotateCenterLayer layer.

Type	Description
String	A String expression that determines the x-origin of the rotation point relative to the <a href="#">RotateCenterLayer</a> layer.

By default, the RotateCenterX / [RotateCenterY](#) property is empty. If the RotateCenterX / [RotateCenterY](#) property is empty, missing or invalid, the center of the layer ([RotateCenterLayer](#) layer) is considered the rotation center. Use the [DefaultLayer\(exDefLayerRotateCenterX\)](#) property to specify the default value for the RotateCenterX property, before adding the layer.

You can change the [Debug](#) property ( to exDebugLayerDragRotate, for instance ) of the [DragInfo](#) object during the [DragStart](#) event to show the current rotation point as shown in the following screen shot:



For instance:

- "lwidth/2 + 78", defines the center 78 pixels to the right relative to the center of the layer.

The RotateCenterX property supports the following keywords:

- **lwidth** keyword, indicates the width in pixels of the layer
- **width** keyword, specifies the width in pixels of the view / control
- **lheight** keyword, indicates the height in pixels of the layer

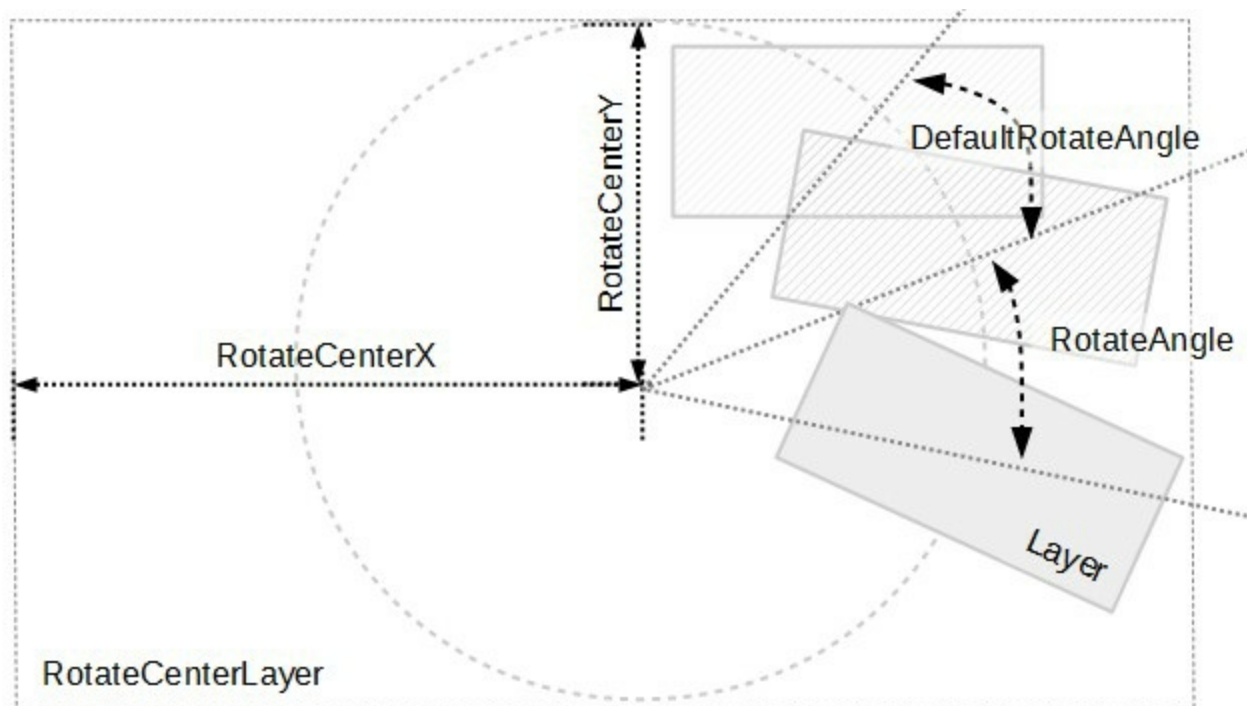
- **height** keyword, specifies the height in pixels of the view / control

Also, this property supports all constants, operators and functions defined [here](#).

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is `exDoRotamove`:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.

- [RotamoveOffsetY](#), specifies the y-offset of the layer.

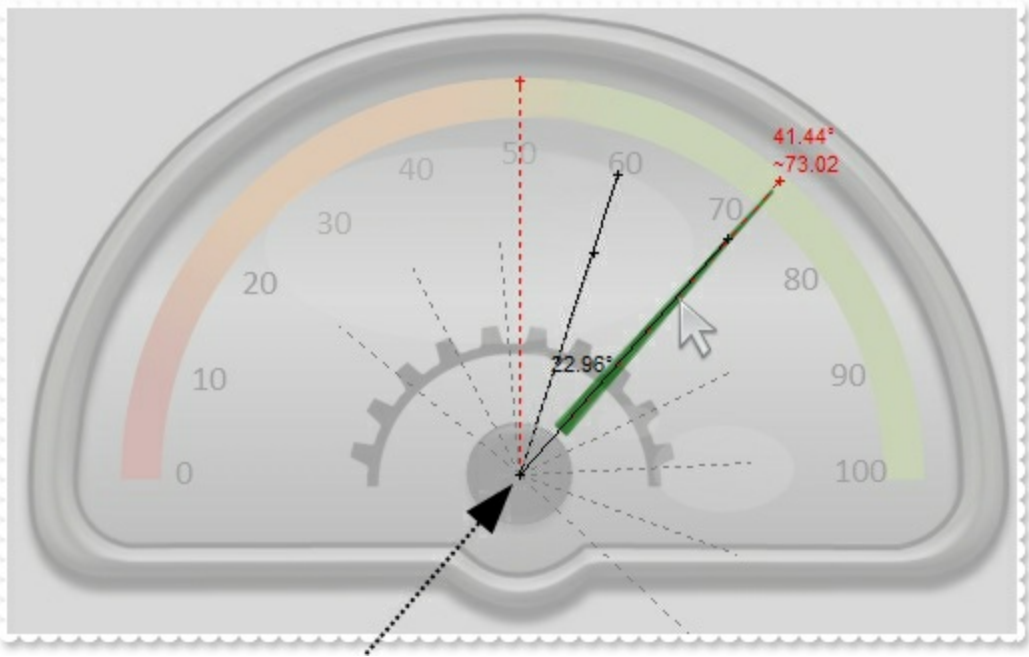
# property Layer.RotateCenterY as String

Indicates the expression that determines the y-origin of the rotation point relative to the RotateCenterLayer layer.

Type	Description
String	A String expression that determines the y-origin of the rotation point relative to the <a href="#">RotateCenterLayer</a> layer.

By default, the [RotateCenterX](#) / RotateCenterY property is empty. If the [RotateCenterX](#) / RotateCenterY property is empty, missing or invalid, the center of the layer ( [RotateCenterLayer](#) layer) is considered the rotation center. Use the [DefaultLayer\(exDefLayerRotateCenterX\)](#) property to specify the default value for the RotateCenterX property, before adding the layer.

You can change the [Debug](#) property ( to exDebugLayerDragRotate, for instance ) of the [DragInfo](#) object during the [DragStart](#) event to show the current rotation point as shown in the following screen shot:



For instance:

- "lwidth/2 + 78", defines the center 78 pixels to the right relative to the center of the layer.

The RotateCenterX property supports the following keywords:

- **lwidth** keyword, indicates the width in pixels of the layer
- **width** keyword, specifies the width in pixels of the view / control
- **lheight** keyword, indicates the height in pixels of the layer



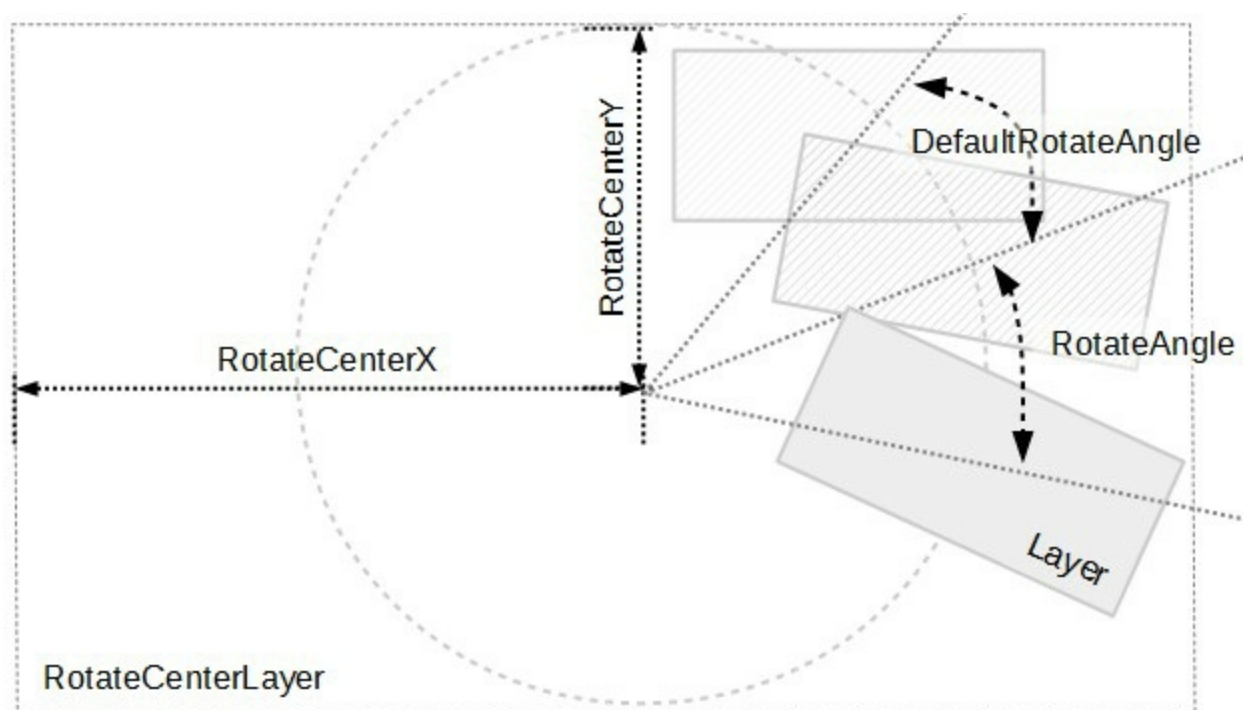
- **height** keyword, specifies the height in pixels of the view / control

Also, this property supports all constants, operators and functions defined [here](#).

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the RotateCenterLayer layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the RotateCenterLayer layer.

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.



- [RotamoveOffsetY](#), specifies the y-offset of the layer.

# property Layer.RotateClip as Boolean

Specifies whether the layer's clipping region is rotated once the layer is rotated.

Type	Description
Boolean	A Boolean expression that specifies whether the layer's clipping region is rotated once the current later is rotated.

By default, the RotateClip property is False, which indicates that has no effect. The RotateClip property specifies whether the layer's clipping region is rotated once the layer is rotated. The [Clip](#) property gets access to the layer's clipping object. Use the [DefaultLayer\(exDefLayerRotateClip\)](#) property to specify the default value for the RotateClip property, before adding the layer. The [OnDrag](#) property indicates the action to be performed when the user drags the layer. The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable.

The RotateClip property has effect if:

- The RotateClip property is True.
- The current layer is rotated
- The layer's [Clip](#) property clips the current layer.

Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.

- [RotamoveOffsetY](#), specifies the y-offset of the layer.

# property Layer.RotateType as RotateTypeEnum

Returns or sets a value that indicates whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ...

Type	Description
<a href="#">RotateTypeEnum</a>	A RotateTypeEnum expression that specifies the method the rotation of the layer is performed.

By default, the RotateType property is exRotateFast. The RotateType property specifies whether the layer's rotation is performed fast, by shearing ( high quality rotation ), ... Use the [DefaultLayer\(exDefLayerDefaultRotateType\)](#) property to specify the default value for the RotateType property, before adding any layer.

The following screen shot shows the hands of the clock, while the RotateType property is **exRotateBilinearInterpolation**:



The following screen shot shows the hands of the clock, while the RotateType property is **exRotateFast** (by default):



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is exDoRotamove:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.



# property Layer.Selectable as Boolean

Returns or sets a value that indicates whether the layer is selectable.

Type	Description
Boolean	A Boolean expression that specifies whether the layer is selectable or unselectable.

By default, the Selectable property is True, which indicates that the layer is selectable. The Selectable property returns or sets a value that indicates whether the layer is selectable. Use the [DefaultLayer\(exDefLayerSelectable\)](#) property to specify the default value for the Selectable property, before adding any layer. The [Visible](#) property shows or hides a specific layer (visible). The [LayerFromPoint](#) property retrieves the layer from point that's visible and selectable. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( dragable ). You can use the [Grayscale](#) property to show the entire layer in gray scale ( disable state). For instance, you can simulate a disabled layer by changing the layer's Grayscale property on True, and setting the layer's Selectable property on False.

# property Layer.ShowHandCursor as Boolean

Returns or sets a value that indicates whether the hand cursor is shown when it hovers a visible / selectable / draggable layer.

Type	Description
Boolean	A Boolean expression that indicates whether the hand cursor is shown when it hovers a visible / selectable / draggable layer.

By default, the ShowHandCursor property is True, which indicates that the hand cursor is shown over any layer that has:

- [Visible](#) property on True,
- [Selectable](#) property on True,
- [OnDrag](#) property is not exDoNothing

The [Visible](#) property shows or hides a specific layer (visible). The [Position](#) property specifies the position of the layer, in the layers collection. The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [LayerFromPoint](#) property retrieves the layer from point that's visible and selectable. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ).



# property Layer.ToolTip as String

Gets or sets a value (tooltip) that's displayed once the cursor hovers the layer.

Type	Description
String	A String expression that defines the layer's HTML tooltip that's displayed when the cursor hovers the layer.

By default, the ToolTip / ToolTipTitle property is empty, which indicates no tooltip. The ToolTip gets or sets a value (tooltip) that's displayed once the cursor hovers the layer. The ToolTipTitle property indicates the title of the layer's tooltip. The layer's tooltip is shown if any of the ToolTip / ToolTipTitle property is not empty. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipFont property to change the tooltip's font. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the ShowToolTip method to display a custom tooltip. The DefaultLayer(exDefLayerToolTip) property specifies the default value of the ToolTip property.

The following screen shot shows the current value as a tooltip :



The ToolTip supports the following built-in HTML format:

- **<b> ... </b>** displays the text in **bold**

- `<i> ... </i>` displays the text in *italics*
- `<u> ... </u>` underlines the text
- `<s> ... </s>` Strike-through text
- `<a id;options> ... </a>` displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The `<a>` element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- `<font face;size> ... </font>` displays portions of text with a different font and/or different size. For instance, the "`<font Tahoma;12>bit</font>`" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "`<font ;12>bit</font>`" displays the bit text using the current font, but with a different size.
- `<fgcolor rrggbb> ... </fgcolor>` or `<fgcolor=rrggb> ... </fgcolor>` displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<bgcolor rrggbb> ... </bgcolor>` or `<bgcolor=rrggb> ... </bgcolor>` displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<solidline rrggbb> ... </solidline>` or `<solidline=rrggb> ... </solidline>` draws a solid-line on the bottom side of the current text-line, of specified RGB color. The `<solidline> ... </solidline>` draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<dotline rrggbb> ... </dotline>` or `<dotline=rrggb> ... </dotline>` draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- `<upline> ... </upline>` draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- `<r>` right aligns the text
- `<c>` centers the text
- `<br>` forces a line-break
- `<img>number[:width]</img>` inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.

- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&quot;**; ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **<b>bold</b>** in HTML caption you can use **&lt;b>&gt;bold&lt;/b>**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The **<font>** HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The **<gra>** with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "**<font ;18><gra FFFFFFFF;1;1>**gradient-center**</gra></font>**" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or **<fgcolor>** defines the color to show the inside text. The **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><out 000000><fgcolor=FFFFFF>**outlined**</fgcolor></out></font>**" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or **<fgcolor>** defines the color to show the inside text. The **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><sha>**shadow**</sha></font>**" generates the following picture:

# shadow

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

## outline anti-aliasing

# property Layer.ToolTipTitle as String

Gets or sets a value (title) that's displayed once the cursor hovers the layer.

Type	Description
String	A String expression that defines the title of the layer's tooltip. The title does not support HTML format.

By default, the [ToolTip](#) / ToolTipTitle property is empty, which indicates no tooltip. The ToolTipTitle property indicates the title of the layer's tooltip. The layer's tooltip is shown if any of the [ToolTip](#) / ToolTipTitle property is not empty. Use the [ToolTipWidth](#) property to specify the width of the tooltip window Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. Use the [ShowToolTip](#) method to display a custom tooltip. The [DefaultLayer](#)(exDefLayerToolTipTitl) property specifies the default value of the ToolTipTitle property.

The following screen shot shows the current value as a tooltip :



## property Layer.Top as String

Specifies the expression relative to the view, to determine the y-position to show the current layer on the control.

Type	Description
String	A String value that indicates the expression relative to the view, to determine the y-position to show the current layer on the control.

By default, the Top property is "0". If the Top property is empty, missing or invalid, it is considered "0". If valid, the value of evaluating the Top property indicates the top position of the layer as shown in the picture bellow. Use the [DefaultLayer\(exDefLayerTop\)](#) property to specify the default value for the Top property, before adding any layer.

For instance:

- "0" indicates the top side of the control's view
- "height / 2", half of the view or center of the control's view
- "height - 64", 64 pixels to the bottom side of the control's view

The Top property supports the following keywords:

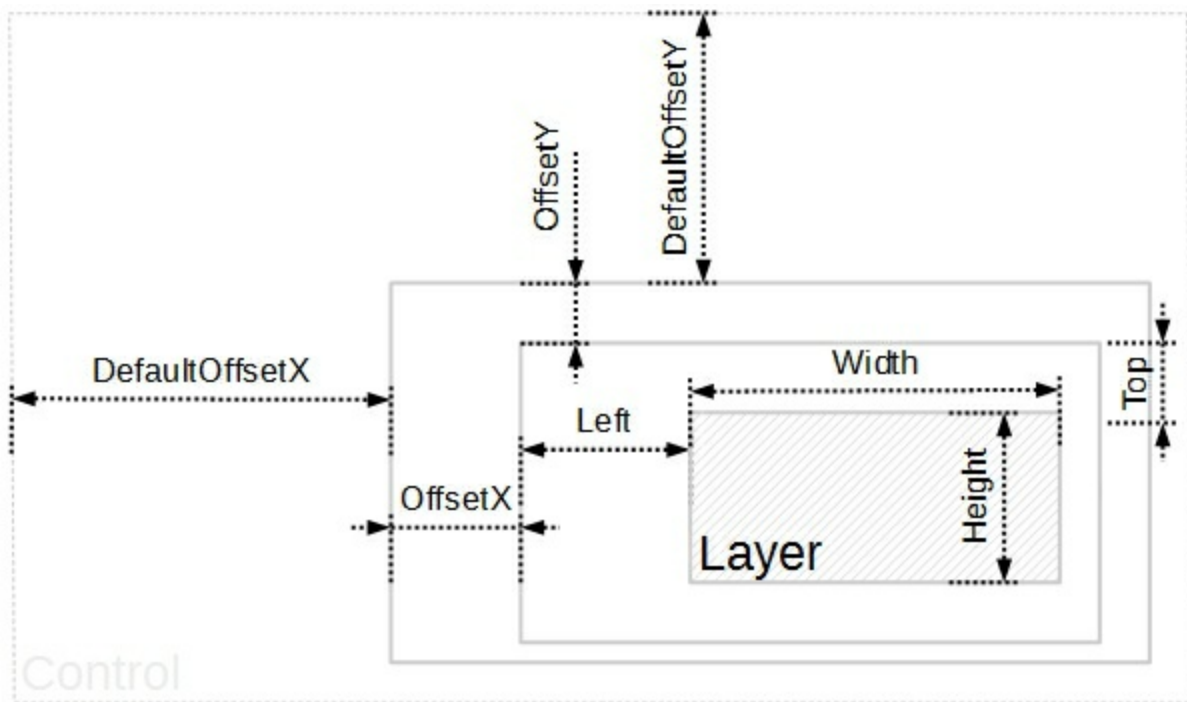
- **width** keyword specifies the width in pixels of the control's view
- **height** keyword specifies the height in pixels of the control's view

Also, this property supports all constants, operators and functions defined [here](#).

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- Top, specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

The following picture shows the position/size properties of the Layer, relative to the view / control:



You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

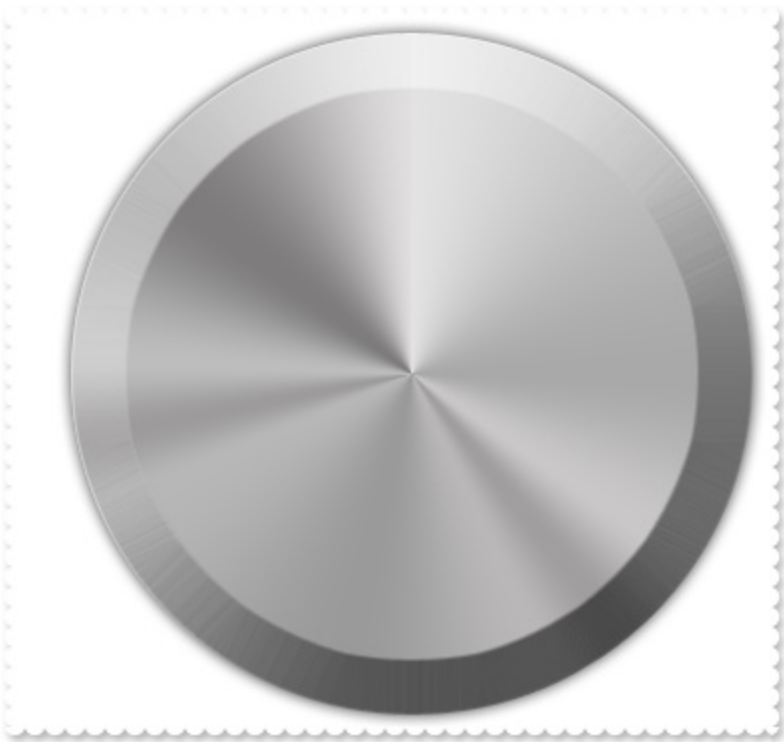
# property Layer.Transparency as Long

Gets or sets a value that indicates percent of the transparency to display the layer.

Type	Description
Long	A Long expression that specifies the percent of layer's transparency.

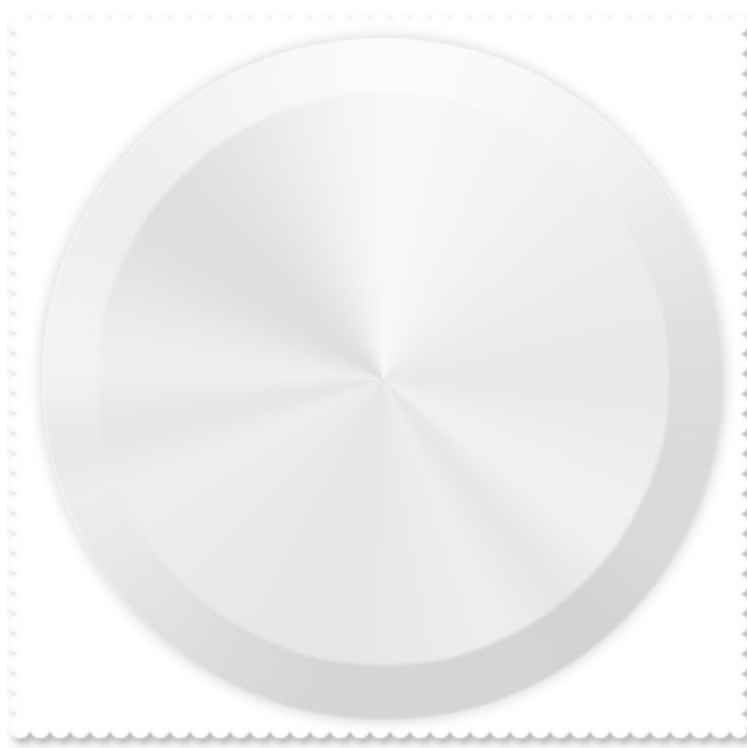
By default, the Transparency 0%, which indicates that no effect is applied to the layer. The Transparency property gets or sets a value that indicates percent of the transparency to display the layer. Use the [DefaultLayer\(exDefLayerTransparency\)](#) property to specify the default value for the Transparency property, before adding any layer.

The following screen shot shows the layer, with Transparency property on 0% ( default ):



The following screen shot shows the layer, with Transparency = 75:





By default, the [AllowSmoothChange](#) property is `exLayerTransparency | exLayerBrightness | exLayerContrast`. Use the `AllowSmoothChange` property to disable changing gradually any brightness / contrast or the transparency, of the layer. For instance, a gradually change means that you can change the layer's transparency from 0 to 50 in a short time, with intermediate values ( smooth change ).

The `AllowSmoothChange` property changes gradually one / or more properties like follow:

- [Brightness](#), Specifies the percent of brightness to apply to the layer.
- [Contrast](#), Specifies the percent of contrast to apply to the layer.
- Transparency, Gets or sets a value that indicates percent of the transparency to display the layer.

The [MouseIn](#) / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects. The `AllowSmoothChange` property specifies the properties of the layers that support smooth change. For instance, you can use the `MouseIn` / `MouseOut` event to change gradually the brightness / contrast or the transparency, of the layer, while the `AllowSmoothChange` property is not `exSmoothChangeless`.

# property Layer.userData as Variant

Indicates any extra data associated with the layer.

Type	Description
Variant	A Variant expression that indicates any extra data associated with the layer.

By default, UserData property is empty. Use the UserData property to associate any extra data to the layer. Use the [DefaultLayer\(exDefLayerUserData\)](#) property to specify the default value for the UserData property, before adding any layer. Use the [UserData](#) of the DragInfo object to associate any extra data to the dragging operation. The [Visible](#) property shows or hides a specific layer (visible). The [Position](#) property specifies the position of the layer, in the layers collection. The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [LayerFromPoint](#) property retrieves the layer from point that's visible and selectable. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ).

# property Layer.Value as Variant

Indicates the object's value.

Type	Description
Variant	A VARIANT expression that specifies the value associated with the layer.

By default, the Value property is empty. The layer's Value could indicate its offset or its rotation angle, based on the OnDrag property. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). Use the [Value](#) property of the Clip object to associate a value with the layer's clipping region. Each layer can associate a value with it, while the control's [Value](#) property can be associated through the [LayerOfValue](#) property with the value of one of the layers within the control.

For instance:

- the control displays a clock, the value could be the current-time
- the control shows a switch, so the value could indicate the state of the switch
- the control shows a thermometer, so the value could be the current temperature
- the control displays a gauge, so the value could be the value on the gauge pointed by the needle

The [Change](#) event occurs when the layer's Value property is changed. During the Change event, you can change values of other layers as well. For instance, if the second-hand of the clock is rotated, you can rotate the hour and the minute-hands of the clock as well. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. *You can call `DragInfo.Debug = -1` during the [DragStart](#) event to display debugging information like current movement, rotation angle when drag operation is performed.*

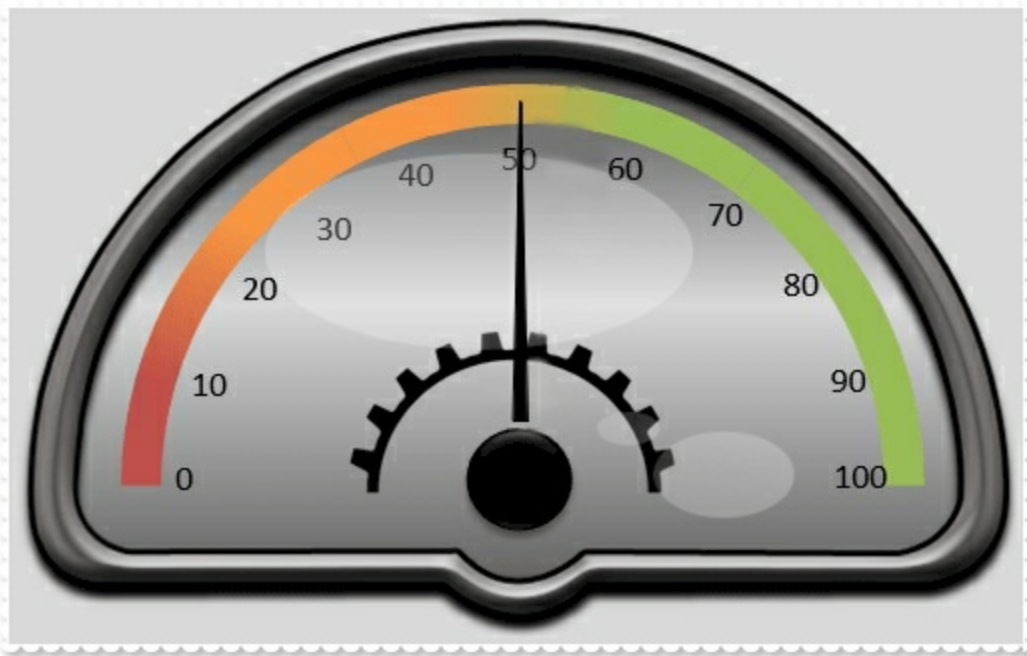
The Value property indicates the **value** keyword in the following properties:

- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the [ValueToOffsetX](#) expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**. The [RotateAngleToValue](#) converts the current rotation angle to a value.

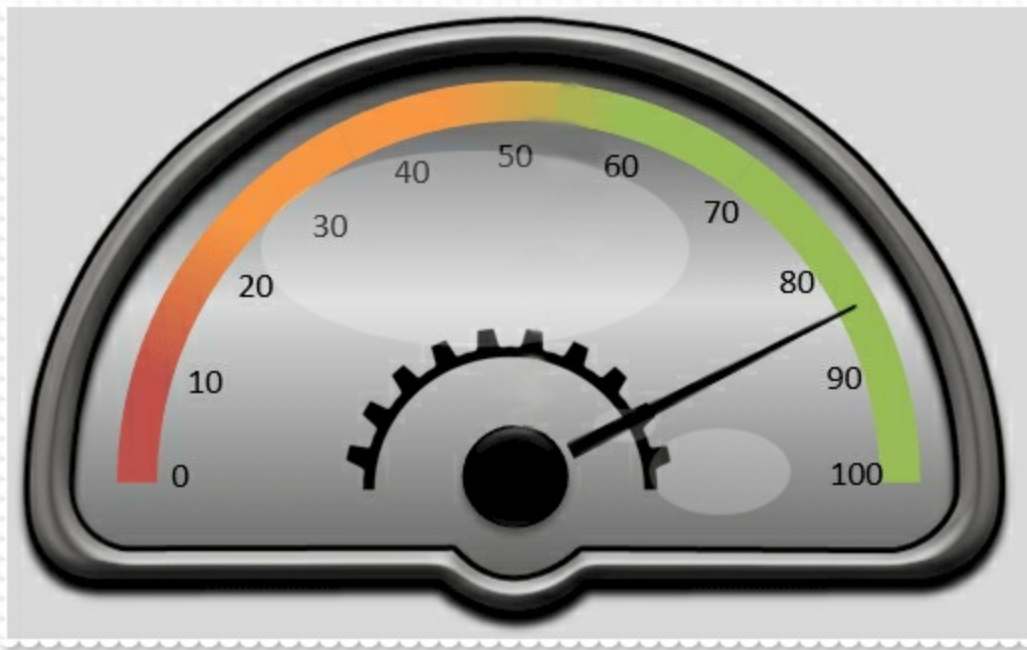
The Value property works in association with the layer's [OnDrag](#) property like follows:

- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetX](#) property indicates the layer's [OffsetX](#) property.
- If the [OnDrag](#) property is **exDoMove**, evaluating the [ValueToOffsetY](#) property indicates the layer's [OffsetY](#) property.
- If the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**, evaluating the [ValueToRotateAngle](#) property indicates the layer's [RotateAngle](#) property.

For instance, having the gauge from the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Guage folder, which includes the background and the needle pictures:



we need to define the value of the needle to be between 0 and 100, so if we call Value property on 85 we should get something like:



In conclusion, what we need to do is:

- defines the "needle" layer as rotate able, using the [OnDrag](#) property
- converts the value of 0-100, to a rotation angle, using the [ValueToRotateAngle](#) property
- converts the rotation angle from 0-360 to the value, using the [RotateAngleToValue](#) property
- limits the rotation angle, using the [RotateAngleValid](#) property

The following samples shows how you can do that:

### VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .DefaultLayer(185) = 2
    .BackColor = RGB(217,217,217)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    With .Layers.Add("background")
        .Background.Picture.Name = "Guage_Background.png"
        .RotateCenterY = "Iheight/2 + 78"
    End With
    With .Layers.Add("needle")
        .Background.Picture.Name = "Guage_Needle.png"
```

```

.OnDrag = 2
.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))"
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
End With
.Value = 85
.EndUpdate
End With

```

## VB6

```

With Gauge1
.BeginUpdate
.DefaultLayer(exDefLayerRotateType) = 2
.BackColor = RGB(217,217,217)
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
With .Layers.Add("background")
.Background.Picture.Name = "Guage_Background.png"
.RotateCenterY = "lheight/2 + 78"
End With
With .Layers.Add("needle")
.Background.Picture.Name = "Guage_Needle.png"
.OnDrag = exDoRotate
.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))"
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
End With
.Value = 85
.EndUpdate
End With

```

## VB.NET

```

With Exgauge1
.BeginUpdate()

```

```

.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateTy

    .BackColor = Color.FromArgb(217,217,217)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    With .Layers.Add("background")
        .Background.Picture.Name = "Guage_Background.png"
        .RotateCenterY = "lheight/2 + 78"
    End With
    With .Layers.Add("needle")
        .Background.Picture.Name = "Guage_Needle.png"
        .OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate
        .RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))"
        .RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
        .ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
    End With
    .Value = 85
    .EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,2)
    .BackColor = RGB(217,217,217)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    With .Layers.Add("background")
        .Background.Picture.Name = "Guage_Background.png"
        .RotateCenterY = "lheight/2 + 78"
    End With
    With .Layers.Add("needle")
        .Background.Picture.Name = "Guage_Needle.png"
        .OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate
    End With
End With

```

```

.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <270 ? 270 :
value ))"
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
End With
.Value = 85
.EndUpdate()
End With

```

## C++

```

/*
Copy and paste the following directives to your header file as
it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
Library'

#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
>GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerRotateType,long(2));
spGauge1->PutBackColor(RGB(217,217,217));
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage");
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->Add("background");
var_Layer->GetBackground()->GetPicture()->PutName("Guage_Background.png");
var_Layer->PutRotateCenterY(L"height/2 + 78");
EXGAUGELib::ILayerPtr var_Layer1 = spGauge1->GetLayers()->Add("needle");
var_Layer1->GetBackground()->GetPicture()->PutName("Guage_Needle.png");
var_Layer1->PutOnDrag(EXGAUGELib::exDoRotate);
var_Layer1->PutRotateAngleValid(L"value < 90 ? value : (value < 180 ? 90 : ( value
< 270 ? 270 : value ))");
var_Layer1->PutRotateAngleToValue(L"value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50");
var_Layer1->PutValueToRotateAngle(L"value < 50 ? (270 + value/50*90) : (value -

```



```
50)/50 * 90");
spGauge1->PutValue(long(85));
spGauge1->EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();
Gauge1-
>DefaultLayer[ExgaugeLib_tlb::DefaultLayerPropertyEnum::exDefLayerRotateType] =
TVariant(2);
Gauge1->BackColor = RGB(217,217,217);
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
ExgaugeLib_tlb::ILayerPtr var_Layer = Gauge1->Layers-
>Add(TVariant("background"));
    var_Layer->Background->Picture->set_Name(TVariant("Guage_Background.png"));
    var_Layer->RotateCenterY = L"height/2 + 78";
ExgaugeLib_tlb::ILayerPtr var_Layer1 = Gauge1->Layers->Add(TVariant("needle"));
    var_Layer1->Background->Picture->set_Name(TVariant("Guage_Needle.png"));
    var_Layer1->OnDrag = ExgaugeLib_tlb::OnDragLayerEnum::exDoRotate;
    var_Layer1->RotateAngleValid = L"value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1->RotateAngleToValue = L"value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1->ValueToRotateAngle = L"value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
Gauge1->set_Value(TVariant(85));
Gauge1->EndUpdate();
```

## C#

```
exgauge1.BeginUpdate();
exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayer

exgauge1.BackColor = Color.FromArgb(217,217,217);
exgauge1.PicturesPath = "C:\\Program
```

```

Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers.Add("background");
    var_Layer.Background.Picture.Name = "Guage_Background.png";
    var_Layer.RotateCenterY = "Iheight/2 + 78";
exontrol.EXGAUGELib.Layer var_Layer1 = exgauge1.Layers.Add("needle");
    var_Layer1.Background.Picture.Name = "Guage_Needle.png";
    var_Layer1.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
exgauge1.Value = 85;
exgauge1.EndUpdate();

```

## JScript/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.DefaultLayer(185) = 2;
    Gauge1.BackColor = 14277081;
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
    var var_Layer = Gauge1.Layers.Add("background");
        var_Layer.Background.Picture.Name = "Guage_Background.png";
        var_Layer.RotateCenterY = "Iheight/2 + 78";
    var var_Layer1 = Gauge1.Layers.Add("needle");
        var_Layer1.Background.Picture.Name = "Guage_Needle.png";
        var_Layer1.OnDrag = 2;

```

```

var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
Gauge1.Value = 85;
Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .DefaultLayer(185) = 2
        .BackColor = RGB(217,217,217)
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
        With .Layers.Add("background")
            .Background.Picture.Name = "Guage_Background.png"
            .RotateCenterY = "lheight/2 + 78"
        End With
        With .Layers.Add("needle")
            .Background.Picture.Name = "Guage_Needle.png"
            .OnDrag = 2
            .RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ?
270 : value ))"
            .RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 +

```

```

50"
    .ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 *
90"
    End With
    .Value = 85
    .EndUpdate
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotate

axGauge1.BackColor = Color.FromArgb(217,217,217);
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
EXGAUGELib.Layer var_Layer = axGauge1.Layers.Add("background");
    var_Layer.Background.Picture.Name = "Guage_Background.png";
    var_Layer.RotateCenterY = "lheight/2 + 78";
EXGAUGELib.Layer var_Layer1 = axGauge1.Layers.Add("needle");
    var_Layer1.Background.Picture.Name = "Guage_Needle.png";
    var_Layer1.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoRotate;
    var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))";
    var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50";
    var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90";
axGauge1.Value = 85;
axGauge1.EndUpdate();

```

```

public void init()
{
    COM com_Background,com_Layer,com_Layer1,com_Picture;
    anytype var_Background,var_Layer,var_Layer1,var_Picture;
    ;

    super();

    exgauge1.BeginUpdate();

    exgauge1.DefaultLayer(185/*exDefLayerRotateType*/,COMVariant::createFromInt(2));
    exgauge1.BackColor(WinApi::RGB2int(217,217,217));
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage");
    var_Layer = COM::createFromObject(exgauge1.Layers()).Add("background");
    com_Layer = var_Layer;
    var_Background = COM::createFromObject(com_Layer.Background());
    com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
    var_Picture;
    com_Picture.Name("Guage_Background.png");
    com_Layer.RotateCenterY("lheight/2 + 78");
    var_Layer1 = COM::createFromObject(exgauge1.Layers()).Add("needle");
    com_Layer1 = var_Layer1;
    var_Background = COM::createFromObject(com_Layer1.Background());
    com_Background = var_Background;
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
    var_Picture;
    com_Picture.Name("Guage_Needle.png");
    com_Layer1.OnDrag(2/*exDoRotate*/);
    com_Layer1.RotateAngleValid("value < 90 ? value : (value < 180 ? 90 : (value <
270 ? 270 : value ))");
    com_Layer1.RotateAngleToValue("value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50");
    com_Layer1.ValueToRotateAngle("value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90");
    exgauge1.Value(COMVariant::createFromInt(85));

```

```
exgauge1.EndUpdate();  
}
```

## Delphi 8 (.NET only)

```
with AxGauge1 do  
begin  
  BeginUpdate();  
  
  set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType, TObj  
  
  BackColor := Color.FromArgb(217,217,217);  
  PicturesPath := 'C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage';  
  with Layers.Add('background') do  
  begin  
    Background.Picture.Name := 'Guage_Background.png';  
    RotateCenterY := 'lheight/2 + 78';  
  end;  
  with Layers.Add('needle') do  
  begin  
    Background.Picture.Name := 'Guage_Needle.png';  
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoRotate;  
    RotateAngleValid := 'value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :  
value ))';  
    RotateAngleToValue := 'value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50';  
    ValueToRotateAngle := 'value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90';  
  end;  
  Value := TObject(85);  
  EndUpdate();  
end
```

## Delphi (standard)

```
with Gauge1 do  
begin  
  BeginUpdate();  
  DefaultLayer[EXGAUGELib_TLB.exDefLayerRotateType] := OleVariant(2);
```

```

BackColor := RGB(217,217,217);
PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage';
with Layers.Add('background') do
begin
    Background.Picture.Name := 'Guage_Background.png';
    RotateCenterY := 'lheight/2 + 78';
end;
with Layers.Add('needle') do
begin
    Background.Picture.Name := 'Guage_Needle.png';
    OnDrag := EXGAUGELib_TLB.exDoRotate;
    RotateAngleValid := 'value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :
value ))';
    RotateAngleToValue := 'value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50';
    ValueToRotateAngle := 'value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90';
end;
Value := OleVariant(85);
EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
.BeginUpdate
.Object.DefaultLayer(185) = 2
.BackColor = RGB(217,217,217)
.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
with .Layers.Add("background")
    .Background.Picture.Name = "Guage_Background.png"
    .RotateCenterY = "lheight/2 + 78"
endwith
with .Layers.Add("needle")
    .Background.Picture.Name = "Guage_Needle.png"
    .OnDrag = 2
    .RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value < 270 ? 270 :

```

```
value))"
```

```
.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 : (value/90)*50 + 50"
```

```
.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value - 50)/50 * 90"
```

```
endwith
```

```
.Value = 85
```

```
.EndUpdate
```

```
endwith
```

## dBASE Plus

```
local oGauge,var_Layer,var_Layer1
```

```
oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
```

```
oGauge.BeginUpdate()
```

```
oGauge.Template = [DefaultLayer(185) = 2] // oGauge.DefaultLayer(185) = 2
```

```
oGauge.BackColor = 0xd9d9d9
```

```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
var_Layer = oGauge.Layers.Add("background")
```

```
var_Layer.Background.Picture.Name = "Guage_Background.png"
```

```
var_Layer.RotateCenterY = "Iheight/2 + 78"
```

```
var_Layer1 = oGauge.Layers.Add("needle")
```

```
var_Layer1.Background.Picture.Name = "Guage_Needle.png"
```

```
var_Layer1.OnDrag = 2
```

```
var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <  
270 ? 270 : value ))"
```

```
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :  
(value/90)*50 + 50"
```

```
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -  
50)/50 * 90"
```

```
oGauge.Value = 85
```

```
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P
```

```
Dim var_Layer as P
```



```
Dim var_Layer1 as P
```

```
oGauge = topparent:CONTROL_ACTIVEX1.activex
```

```
oGauge.BeginUpdate()
```

```
oGauge.Template = "DefaultLayer(185) = 2" // oGauge.DefaultLayer(185) = 2
```

```
oGauge.BackColor = 14277081
```

```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
var_Layer = oGauge.Layers.Add("background")
```

```
var_Layer.Background.Picture.Name = "Guage_Background.png"
```

```
var_Layer.RotateCenterY = "Iheight/2 + 78"
```

```
var_Layer1 = oGauge.Layers.Add("needle")
```

```
var_Layer1.Background.Picture.Name = "Guage_Needle.png"
```

```
var_Layer1.OnDrag = 2
```

```
var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <  
270 ? 270 : value ))"
```

```
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :  
(value/90)*50 + 50"
```

```
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -  
50)/50 * 90"
```

```
oGauge.Value = 85
```

```
oGauge.EndUpdate()
```

## Visual Objects

```
local var_Layer,var_Layer1 as ILayer
```

```
oDCOCX_Exontrol1:BeginUpdate()
```

```
oDCOCX_Exontrol1:[DefaultLayer,exDefLayerRotateType] := 2
```

```
oDCOCX_Exontrol1.BackColor := RGB(217,217,217)
```

```
oDCOCX_Exontrol1.PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
var_Layer := oDCOCX_Exontrol1.Layers.Add("background")
```

```
var_Layer:Background:Picture:Name := "Guage_Background.png"
```

```
var_Layer:RotateCenterY := "Iheight/2 + 78"
```

```
var_Layer1 := oDCOCX_Exontrol1.Layers.Add("needle")
```

```

var_Layer1:Background:Picture:Name := "Guage_Needle.png"
var_Layer1:OnDrag := exDoRotate
var_Layer1:RotateAngleValid := "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))"
var_Layer1:RotateAngleToValue := "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50"
var_Layer1:ValueToRotateAngle := "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90"
oDCOCX_Exontrol1:Value := 85
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge,var_Layer,var_Layer1

oGauge = ole_1.Object
oGauge.BeginUpdate()
oGauge.DefaultLayer(185,2)
oGauge.BackColor = RGB(217,217,217)
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
var_Layer = oGauge.Layers.Add("background")
var_Layer.Background.Picture.Name = "Guage_Background.png"
var_Layer.RotateCenterY = "lheight/2 + 78"
var_Layer1 = oGauge.Layers.Add("needle")
var_Layer1.Background.Picture.Name = "Guage_Needle.png"
var_Layer1.OnDrag = 2
var_Layer1.RotateAngleValid = "value < 90 ? value : (value < 180 ? 90 : ( value <
270 ? 270 : value ))"
var_Layer1.RotateAngleToValue = "value >= 270 ? (value - 270)/90*50 :
(value/90)*50 + 50"
var_Layer1.ValueToRotateAngle = "value < 50 ? (270 + value/50*90) : (value -
50)/50 * 90"
oGauge.Value = 85
oGauge.EndUpdate()

```

## Procedure OnCreate

Forward Send OnCreate

Send ComBeginUpdate

Set ComDefaultLayer OLExDefLayerRotateType to 2

Set ComBackColor to (RGB(217,217,217))

Set ComPicturesPath to "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComAdd of hoLayers "background" to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Variant voBackground

Get ComBackground of hoLayer to voBackground

Handle hoBackground

Get Create (RefClass(cComBackground)) to hoBackground

Set pvComObject of hoBackground to voBackground

Variant voPicture

Get ComPicture of hoBackground to voPicture

Handle hoPicture

Get Create (RefClass(cComPicture)) to hoPicture

Set pvComObject of hoPicture to voPicture

Set ComName of hoPicture to "Guage\_Background.png"

Send Destroy to hoPicture

Send Destroy to hoBackground

Set ComRotateCenterY of hoLayer to "lheight/2 + 78"

Send Destroy to hoLayer

Send Destroy to hoLayers

Variant voLayers1

Get ComLayers to voLayers1

```

Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer1
    Get ComAdd of hoLayers1 "needle" to voLayer1
    Handle hoLayer1
    Get Create (RefClass(cComLayer)) to hoLayer1
    Set pvComObject of hoLayer1 to voLayer1
        Variant voBackground1
        Get ComBackground of hoLayer1 to voBackground1
        Handle hoBackground1
        Get Create (RefClass(cComBackground)) to hoBackground1
        Set pvComObject of hoBackground1 to voBackground1
            Variant voPicture1
            Get ComPicture of hoBackground1 to voPicture1
            Handle hoPicture1
            Get Create (RefClass(cComPicture)) to hoPicture1
            Set pvComObject of hoPicture1 to voPicture1
                Set ComName of hoPicture1 to "Guage_Needle.png"
            Send Destroy to hoPicture1
        Send Destroy to hoBackground1
    Set ComOnDrag of hoLayer1 to OLEexDoRotate
    Set ComRotateAngleValid of hoLayer1 to "value < 90 ? value : (value < 180 ?
90 : ( value < 270 ? 270 : value ))"
    Set ComRotateAngleToValue of hoLayer1 to "value >= 270 ? (value -
270)/90*50 : (value/90)*50 + 50"
    Set ComValueToRotateAngle of hoLayer1 to "value < 50 ? (270 + value/50*90)
: (value - 50)/50 * 90"
    Send Destroy to hoLayer1
Send Destroy to hoLayers1
Set ComValue to 85
Send ComEndUpdate
End_Procedure

```

**XBase++**

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oGauge
```

```
    LOCAL oLayer,oLayer1
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oGauge := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-83ED1790AAD3}*/
```

```
    oGauge:create(,, {10,60},{610,370} )
```

```
        oGauge:BeginUpdate()
```

```
        oGauge:SetProperty("DefaultLayer",185/*exDefLayerRotateType*/,2)
```

```
        oGauge:SetProperty("BackColor",AutomationTranslateColor( GraMakeRGBColor  
( { 217,217,217 } ) , .F. ))
```

```
        oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
```

```
        oLayer := oGauge:Layers():Add("background")
```

```
            oLayer:Background():Picture():Name := "Guage_Background.png"
```

```
            oLayer:RotateCenterY := "lheight/2 + 78"
```

```
        oLayer1 := oGauge:Layers():Add("needle")
```

```
            oLayer1:Background():Picture():Name := "Guage_Needle.png"
```

```
            oLayer1:OnDrag := 2/*exDoRotate*/
```

```
            oLayer1:RotateAngleValid := "value < 90 ? value : (value < 180 ? 90 : ( value <  
270 ? 270 : value ))"
```

```
            oLayer1:RotateAngleToValue := "value >= 270 ? (value - 270)/90*50 :  
(value/90)*50 + 50"
```

```
            oLayer1:ValueToRotateAngle := "value < 50 ? (270 + value/50*90) : (value -  
50)/50 * 90"
```

```
        oGauge:Value := 85
```

```
oGauge:EndUpdate()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

# property Layer.ValueToOffsetX as String

Specifies the expression to convert the value to x-offset.

Type	Description
String	A String value that defines the expression to convert the value to x-offset.

By default, the ValueToOffsetX property is empty. If the ValueToOffsetX property is empty, missing or invalid it has no effect. If the ValueToOffsetX property is valid, the result of evaluation of it, indicates the value of the [OffsetX](#) property, while the [OnDrag](#) property is **exDoMove**. The [OffsetXValid](#) property limits / validates the x-offset value of the layer. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [Change](#) event occurs when the layer's Value property is changed. During the Change event, you can change values of other layers as well. For instance, if the second-hand of the clock is rotated, you can rotate the hour and the minute-hands of the clock as well. The [ValueToOffsetY](#) property specifies the expression to convert the value to y-offset. Use the [DefaultLayer\(exDefLayerValueToOffsetX\)](#) property to specify the default value for the ValueToOffsetX property, before adding the layer.

For instance:

- the control shows a switch, so the value could indicate the state of the switch
- the control shows a thermometer, so the value could be the current temperature

For instance:

- "0", no horizontal move, or assigns always 0 to [OffsetX](#) property
- "value / 2", specifies half of the layer's [Value](#)
- "value = 0 ? 0 : 48", indicates that the [OffsetX](#) property is 0 if the layer's [Value](#) is 0 or 48 if different than 0

The ValueToOffsetX property supports the following keywords:

- **value** keyword indicates the layer's [Value](#) property.

Also, this property supports all constants, operators and functions defined [here](#).

The [Value](#) property indicates the **value** keyword in the following properties:

- ValueToOffsetX, Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the ValueToOffsetX expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.

- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is exDoMove. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is exDoRotate or exDoRotamove. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The Value property works in association with the layer's [OnDrag](#) property like follows:

- If the [OnDrag](#) property is **exDoMove**, evaluating the ValueToOffsetX property indicates the layer's [OffsetX](#) property.
- If the [OnDrag](#) property is exDoMove, evaluating the [ValueToOffsetY](#) property indicates the layer's [OffsetY](#) property.
- If the [OnDrag](#) property is exDoRotate or exDoRotamove, evaluating the [ValueToRotateAngle](#) property indicates the layer's [RotateAngle](#) property.



# property Layer.ValueToOffsetY as String

Specifies the expression to convert the value to y-offset.

Type	Description
String	A String value that defines the expression to convert the value to y-offset.

By default, the ValueToOffsetY property is empty. If the ValueToOffsetY property is empty, missing or invalid it has no effect. If the ValueToOffsetY property is valid, the result of evaluation of it, indicates the value of the [OffsetY](#) property, while the [OnDrag](#) property is **exDoMove**. The [OffsetYValid](#) property limits / validates the y-offset value of the layer. The [OffsetX](#) / [OffsetY](#) property specifies the (x,y)-position of the layer, relative to the upper-left corner of the control. The [Change](#) event occurs when the layer's Value property is changed. During the Change event, you can change values of other layers as well. For instance, if the second-hand of the clock is rotated, you can rotate the hour and the minute-hands of the clock as well. The [ValueToOffsetX](#) property specifies the expression to convert the value to x-offset. Use the [DefaultLayer\(exDefLayerValueToOffsetY\)](#) property to specify the default value for the ValueToOffsetY property, before adding the layer.

For instance:

- the control shows a switch, so the value could indicate the state of the switch
- the control shows a thermometer, so the value could be the current temperature

For instance:

- "0", no vertical move, or assigns always 0 to [OffsetY](#) property
- "value / 2", specifies half of the layer's [Value](#)
- "value = 0 ? 0 : 48", indicates that the [OffsetY](#) property is 0 if the layer's [Value](#) is 0 or 48 if different than 0

The ValueToOffsetY property supports the following keywords:

- **value** keyword indicates the layer's [Value](#) property.

Also, this property supports all constants, operators and functions defined [here](#).

The [Value](#) property indicates the **value** keyword in the following properties:

- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the [ValueToOffsetX](#) expression, while the [OnDrag](#) property is exDoMove. The [OffsetToValue](#) converts the current offset to a value.
- ValueToOffsetY, Specifies the expression to convert the value to y-offset. The layer's

[OffsetY](#) property is the result of evaluating the ValueToOffsetY expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.

- [ValueToRotateAngle](#), Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is exDoRotate or exDoRotamove. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The Value property works in association with the layer's [OnDrag](#) property like follows:

- If the [OnDrag](#) property is exDoMove, evaluating the [ValueToOffsetX](#) property indicates the layer's [OffsetX](#) property.
- If the [OnDrag](#) property is **exDoMove**, evaluating the ValueToOffsetY property indicates the layer's [OffsetY](#) property.
- If the [OnDrag](#) property is exDoRotate or exDoRotamove, evaluating the [ValueToRotateAngle](#) property indicates the layer's [RotateAngle](#) property.

# property Layer.ValueToRotateAngle as String

Specifies the expression to convert the value to rotating angle

Type	Description
String	A String value that defines the expression to convert the value to rotating angle

By default, the ValueToRotateAngle property is empty. If the ValueToRotateAngle property is empty, missing or invalid it has no effect. If the ValueToRotateAngle property is valid, the result of evaluation of it, indicates the value of the [RotateAngle](#) property, while the [OnDrag](#) property is **exDoRotate** or **exDoRotamove**. The [RotateAngleValid](#) property limits / validates the rotation angle of the layer. The [RotateAngle](#) property specifies the angle to rotate the layer. The [Change](#) event occurs when the layer's Value property is changed. During the Change event, you can change values of other layers as well. For instance, if the second-hand of the clock is rotated, you can rotate the hour and the minute-hands of the clock as well. Use the [DefaultLayer\(exDefLayerValueToRotateAngle\)](#) property to specify the default value for the ValueToRotateAngle property, before adding the layer. The [RotateAngleToValue](#) converts the current rotation angle to a value.

For instance:

- the control displays a clock, the value could be the current-time
- the control displays a gauge, so the value could be the value on the gauge pointed by the needle

For instance:

- "0", no rotation, or assigns always 0 to [RotateAngle](#) property
- "value / 2", specifies half of the layer's [Value](#)
- "value / 100 \* 360", [Value](#) percent from 360 degree. For instance, if the value is 50, the expression returns 180, and if 100, the expression returns 360
- "value < 50 ? (270 + value/50\*90) : (value - 50)/50 \* 90", for a value less than 50 returns the angle between 270 and 360, and for a value grater than 50, from 0 to 90
- "2 \* 360 \* ( (0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) )", indicates the position ( rotation angle ) of the clock's hours-hand giving the time value.
- "((1:=( ( 0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) \* 24 )) - floor(=:1)) \* 360", indicates the position ( rotation angle ) of the clock's minutes-hand giving the time value.
- "((2:=( ((1:=( ( 0:=(value < 0 ? floor(value) + 1 - value : value - floor(value))) < 0.5 ? =:0 : (0:= (=:0 - 0.5)) ) \* 24 )) - floor(=:1)) \* 60 )) - floor(=:2)) \* 360", indicates the position ( rotation angle ) of the clock's seconds-hand giving the time value.

The `ValueToRotateAngle` property supports the following keywords:

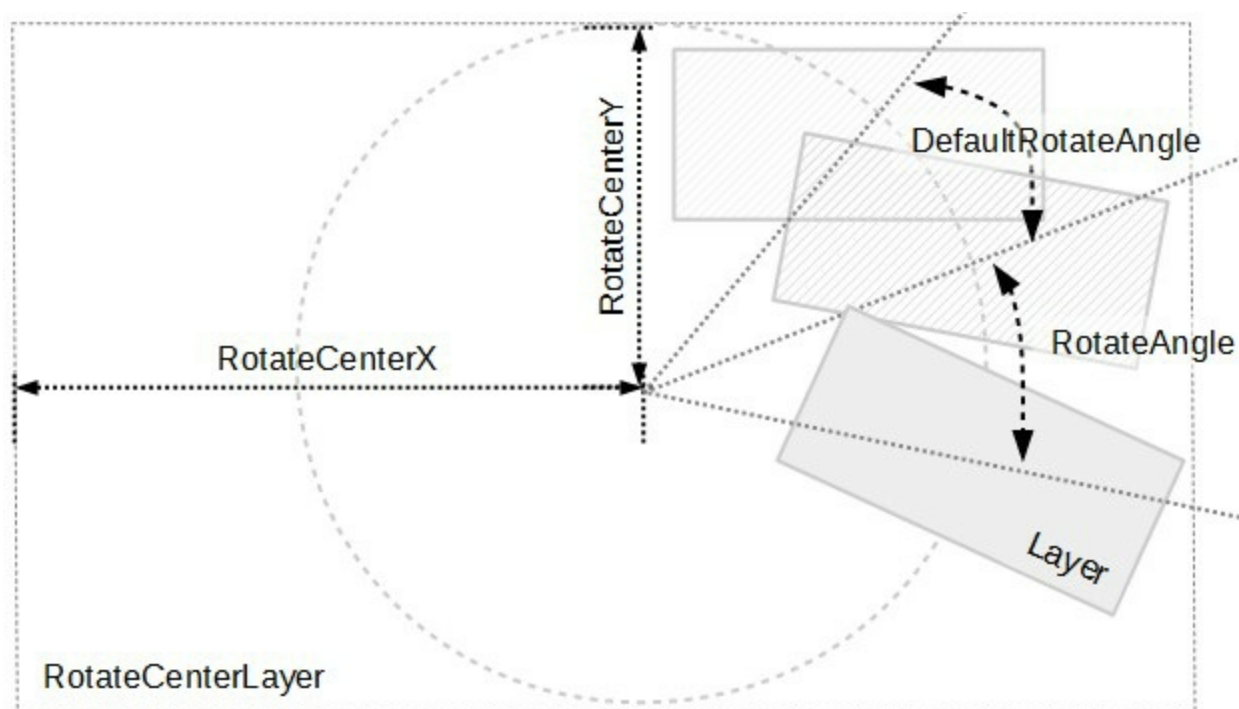
- **value** keyword indicates the layer's [Value](#) property.

Also, this property supports all constants, operators and functions defined [here](#).

The [Value](#) property indicates the **value** keyword in the following properties:

- `ValueToRotateAngle`, Specifies the expression to convert the value to rotating angle. The layer's [RotateAngle](#) property is the result of evaluating the [ValueToRotateAngle](#) expression, while the [OnDrag](#) property is `exDoRotate` or `exDoRotamove`. The [RotateAngleToValue](#) converts the current rotation angle to a value.
- [ValueToOffsetX](#), Specifies the expression to convert the value to x-offset. The layer's [OffsetX](#) property is the result of evaluating the `ValueToOffsetX` expression, while the [OnDrag](#) property is **exDoMove**. The [OffsetToValue](#) converts the current offset to a value.
- [ValueToOffsetY](#), Specifies the expression to convert the value to y-offset. The layer's [OffsetY](#) property is the result of evaluating the [ValueToOffsetY](#) expression, while the [OnDrag](#) property is `exDoMove`. The [OffsetToValue](#) converts the current offset to a value.

The following picture shows the rotation properties of the Layer, relative to the [RotateCenterLayer](#) layer:



Any of the following properties can be used to rotate the layer:

- [DefaultRotateAngle](#), specifies the default angle to rotate the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.

- [RotateAngleValid](#), validates / limits the rotation angle of the layer.
- [Value](#) and [ValueToRotateAngle](#), specifies the expression to convert the value to rotating angle. The [RotateAngleToValue](#) converts the current rotation angle to a value.

The following properties can be used to specify a different rotation center:

- [RotateCenterLayer](#), indicates the index of the layer the rotation is around.
- [RotateCenterX](#), indicates the expression that determines the x-origin of the rotation point relative to the [RotateCenterLayer](#) layer.
- [RotateCenterY](#), indicates the expression that determines the y-origin of the rotation point relative to the [RotateCenterLayer](#) layer.

The following properties can be used to change the rotation center, while the layer's [OnDrag](#) property is `exDoRotamove`:

- [RotamoveCenterX](#), specifies the x-position of the layer's center.
- [RotamoveCenterY](#), specifies the y-position of the layer's center.
- [RotamoveOffsetX](#), specifies the x-offset of the layer.
- [RotamoveOffsetY](#), specifies the y-offset of the layer.

## property Layer.Visible as Boolean

Retrieves or sets a value indicating whether the layer is visible or hidden.

Type	Description
Boolean	A Boolean expression that specifies whether the layer is visible or hidden.

By default, the Visible property is True, so the layer is visible. The Visible property shows or hides a specific layer (visible). Use the [DefaultLayer\(exDefLayerVisible\)](#) property to specify the default value for the Visible property, before adding any layer. The [Position](#) property specifies the position of the layer, in the layers collection. The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [LayerFromPoint](#) property retrieves the layer from point that's visible and selectable. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( dragable ). The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The VisibleCount property indicates the number of visible layers within the control. The [ShowLayers](#) property indicates the only layers to be shown on the control.

## property Layer.Width as String

Specifies the expression relative to the view, to determine the width to show the current layer on the control.

Type	Description
String	A String value that specifies the expression relative to the view, to determine the width to show the current layer on the control.

By default, the Width property is "width". If the Width property is empty, missing or invalid, it is considered "width". If valid, the value of evaluating the Width property indicates the width of the layer as shown in the picture bellow. Use the [DefaultLayer\(exDefLayerWidth\)](#) property to specify the default value for the Width property, before adding any layer. The [LayerAutoSize](#) property resizes all layers based on the picture of the first layer.

For instance:

- "0" indicates that the layer's width is 0
- "width / 2", half of the view or center of the control's view
- "width - 64", 64 pixels to the right side of the control's view

The Width property supports the following keywords:

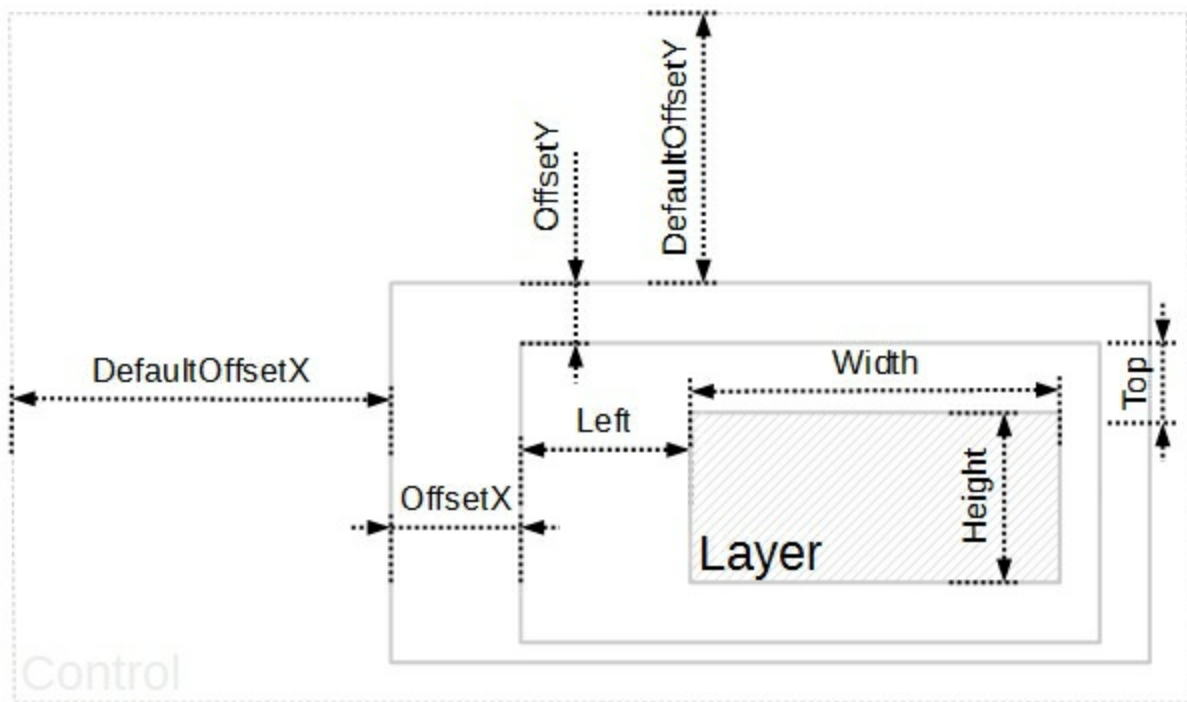
- **width** keyword specifies the width in pixels of the control's view
- **height** keyword specifies the height in pixels of the control's view

Also, this property supports all constants, operators and functions defined [here](#).

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- Width, specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

The following picture shows the position/size properties of the Layer, relative to the view / control:



You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [DefaultOffsetX](#), gets or sets a value that indicates the default x-offset of the layer.
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetXValid](#), validates the x-offset value of the layer.
- [Value](#) and [ValueToOffsetX](#) specifies the expression to convert the value to x-offset.
- [DefaultOffsetY](#), gets or sets a value that indicates the default y-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.
- [OffsetYValid](#), validates the y-offset value of the layer.
- [Value](#) and [ValueToOffsetY](#) specifies the expression to convert the value to y-offset.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.



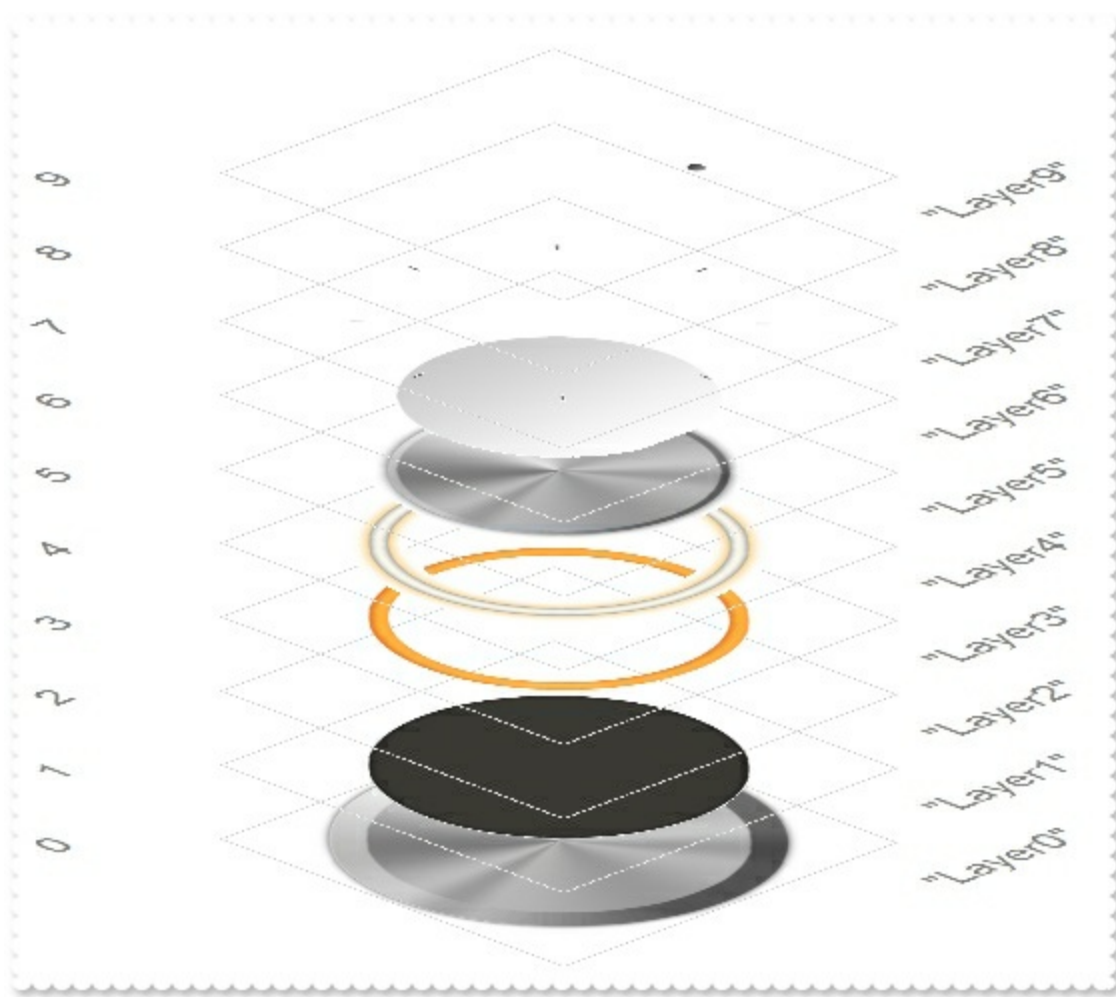
# Layers object

The Layers object holds a collection of [Layer](#) objects. The [Layers](#) property gives access to the control's Layers collection. Any layer can display unlimited opaque / transparent graphics, HTML text, can be visible, selectable, draggable and so on. Any layer can change its position in the layers collection as well. The Layer can change its brightness, contrast, grayscale or transparency as well.

The following screen show shows a control with 10 layers:



while the following screen shot shows how the layers are arranged:



The Layers collection supports the following properties and methods.

Name	Description
<a href="#">Add</a>	Adds a Layer object to the collection and returns a reference to the newly created object.
<a href="#">Clear</a>	Removes all objects in a collection.
<a href="#">Count</a>	Specifies the number of layers.
<a href="#">Item</a>	Returns a specific Layer of the Layers collection.
<a href="#">Remove</a>	Removes a specific member from the Layers collection.
<a href="#">VisibleCount</a>	Specifies the number of visible layers.
<a href="#">VisibleItem</a>	Returns the visible Layer of the Layers collection, based on its position.

## Method **Layers.Add ([Key as Variant])**

Adds a Layer object to the collection and returns a reference to the newly created object.

Type	Description
Key as Variant	[Optional] A VARIANT expression that indicates the key to identify the newly create layer.
Return	Description
Layer	A <a href="#">Layer</a> object that indicates the newly created layer.

The Add method adds a new layer to the control. The [Count](#) property specifies the number of layers in the control. The [PicturesPath](#) Specifies the path to load the pictures from. The [PicturesName](#) property specifies the expression that indicates the name of the picture to be loaded on each layer. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index Count - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers collection. The [ShowLayers](#) property indicates the only layers to be shown on the control.

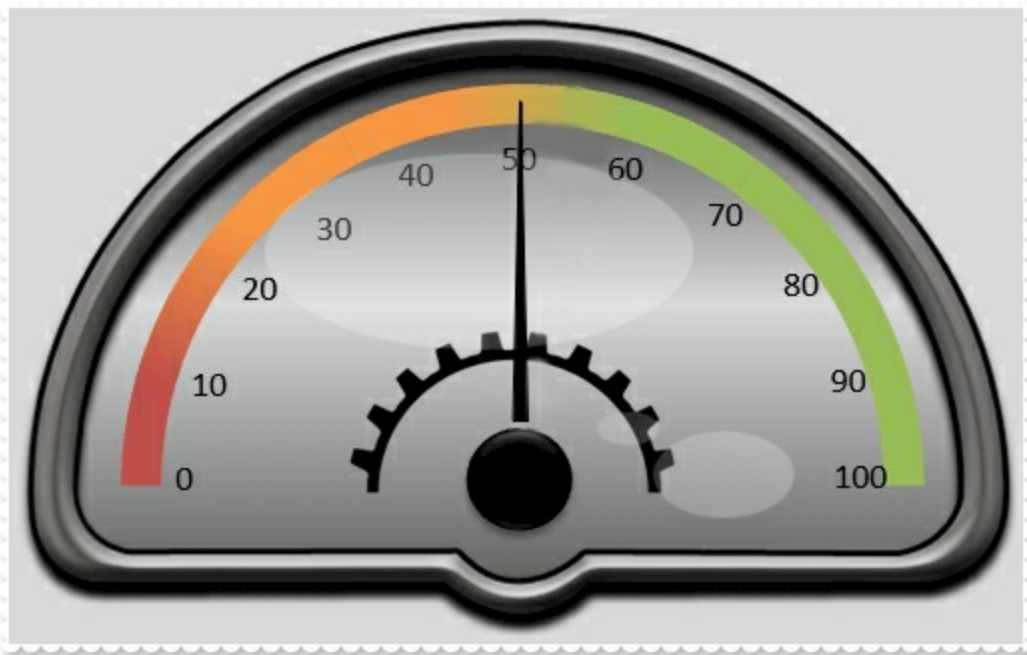
The following properties can be used to add / remove layers within the control:

- Count property, adds / removes layers to / from the control
- [Add](#) method, adds a new layer to the control.

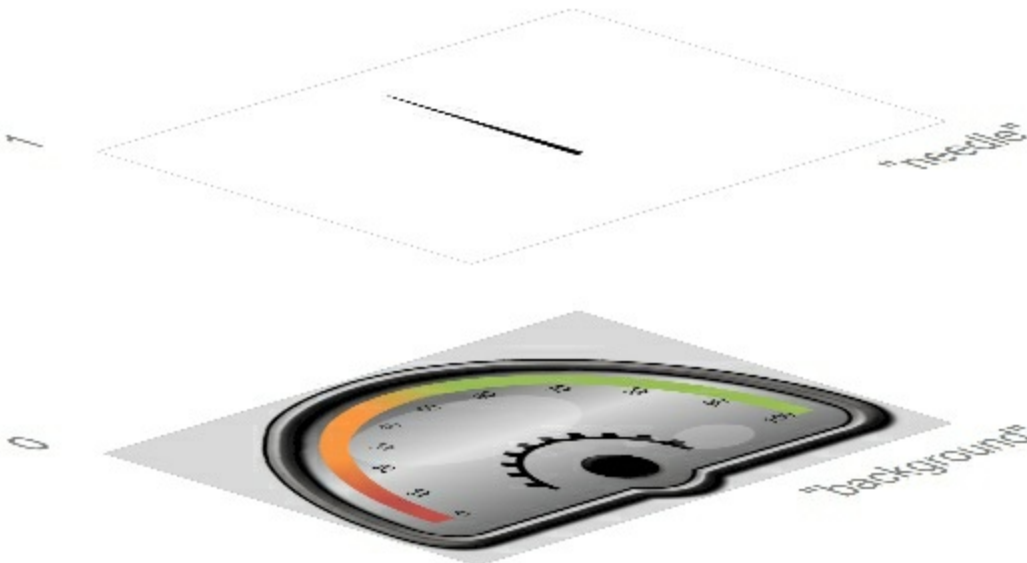
The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following sample creates a view from two pictures: "[Guage\\_Background.png](#)" and "[Guage\\_Needle.png](#)" from the C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Guage.



The sample shows how to add two new layers with the keys: "background" and "needle"



## VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
    .Layers.Add("background").Background.Picture.Name = "Guage_Background.png"
    .Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png"
    .EndUpdate
End With
```

With Gauge1

.BeginUpdate

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"

.Layers.Add("background").Background.Picture.Name = "Guage\_Background.png"

.Layers.Add("needle").Background.Picture.Name = "Guage\_Needle.png"

.EndUpdate

End With

## VB.NET

With Exgauge1

.BeginUpdate()

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"

.Layers.Add("background").Background.Picture.Name = "Guage\_Background.png"

.Layers.Add("needle").Background.Picture.Name = "Guage\_Needle.png"

.EndUpdate()

End With

## VB.NET for /COM

With AxGauge1

.BeginUpdate()

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"

.Layers.Add("background").Background.Picture.Name = "Guage\_Background.png"

.Layers.Add("needle").Background.Picture.Name = "Guage\_Needle.png"

.EndUpdate()

End With

## C++

/\*

*Copy and paste the following directives to your header file as  
it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control  
Library'*

```

#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage");
spGauge1->GetLayers()->Add("background")->GetBackground()->GetPicture()-
> PutName("Guage_Background.png");
spGauge1->GetLayers()->Add("needle")->GetBackground()->GetPicture()-
> PutName("Guage_Needle.png");
spGauge1->EndUpdate();

```

## C++ Builder

```

Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
Gauge1->Layers->Add(TVariant("background"))->Background->Picture-
> set_Name(TVariant("Guage_Background.png"));
Gauge1->Layers->Add(TVariant("needle"))->Background->Picture-
> set_Name(TVariant("Guage_Needle.png"));
Gauge1->EndUpdate();

```

## C#

```

exgauge1.BeginUpdate();
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
exgauge1.Layers.Add("background").Background.Picture.Name =
"Guage_Background.png";
exgauge1.Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png";
exgauge1.EndUpdate();

```

## JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";
    Gauge1.Layers.Add("background").Background.Picture.Name =
"Gauge_Background.png";
    Gauge1.Layers.Add("needle").Background.Picture.Name = "Gauge_Needle.png";
    Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>
```

## VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage"
        .Layers.Add("background").Background.Picture.Name =
"Gauge_Background.png"
        .Layers.Add("needle").Background.Picture.Name = "Gauge_Needle.png"
        .EndUpdate
    End With
End Function
</SCRIPT>
</BODY>
```

End Function

</SCRIPT>

</BODY>

## C# for /COM

```
axGauge1.BeginUpdate();  
axGauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage";  
axGauge1.Layers.Add("background").Background.Picture.Name =  
"Guage_Background.png";  
axGauge1.Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png";  
axGauge1.EndUpdate();
```

## X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Background,com_Layer,com_Picture;  
    anytype var_Background,var_Layer,var_Picture;  
    ;  
  
    super();  
  
    exgauge1.BeginUpdate();  
    exgauge1.PicturesPath("C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Guage");  
    var_Layer = COM::createFromObject(exgauge1.Layers()).Add("background");  
    com_Layer = var_Layer;  
    var_Background = COM::createFromObject(com_Layer).Background();  
    com_Background = var_Background;  
    var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =  
    var_Picture;  
    com_Picture.Name("Guage_Background.png");  
    var_Layer = COM::createFromObject(exgauge1.Layers()).Add("needle"); com_Layer  
    = var_Layer;
```



```

var_Background = COM::createFromObject(com_Layer).Background();
com_Background = var_Background;
var_Picture = COM::createFromObject(com_Background).Picture(); com_Picture =
var_Picture;
com_Picture.Name("Guage_Needle.png");
exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage';
  Layers.Add('background').Background.Picture.Name := 'Guage_Background.png';
  Layers.Add('needle').Background.Picture.Name := 'Guage_Needle.png';
  EndUpdate();
end

```

## Delphi (standard)

```

with Gauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage';
  Layers.Add('background').Background.Picture.Name := 'Guage_Background.png';
  Layers.Add('needle').Background.Picture.Name := 'Guage_Needle.png';
  EndUpdate();
end

```

## VFP

```

with thisform.Gauge1
  .BeginUpdate
  .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"

```

```
.Layers.Add("background").Background.Picture.Name = "Guage_Background.png"  
.Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png"  
.EndUpdate  
endwith
```

## dBASE Plus

```
local oGauge,var_Picture,var_Picture1  
  
oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"  
// oGauge.Layers.Add("background").Background.Picture.Name =  
"Guage_Background.png"  
var_Picture = oGauge.Layers.Add("background").Background.Picture  
with (oGauge)  
    TemplateDef = [dim var_Picture]  
    TemplateDef = var_Picture  
    Template = [var_Picture.Name = "Guage_Background.png"]  
endwith  
// oGauge.Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png"  
var_Picture1 = oGauge.Layers.Add("needle").Background.Picture  
with (oGauge)  
    TemplateDef = [dim var_Picture1]  
    TemplateDef = var_Picture1  
    Template = [var_Picture1.Name = "Guage_Needle.png"]  
endwith  
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P  
Dim var_Picture as local  
Dim var_Picture1 as local  
  
oGauge = topparent:CONTROL_ACTIVEX1.activex
```

```

oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
' oGauge.Layers.Add("background").Background.Picture.Name =
"Guage_Background.png"
var_Picture = oGauge.Layers.Add("background").Background.Picture
oGauge.TemplateDef = "dim var_Picture"
oGauge.TemplateDef = var_Picture
oGauge.Template = "var_Picture.Name = `Guage_Background.png`"

' oGauge.Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png"
var_Picture1 = oGauge.Layers.Add("needle").Background.Picture
oGauge.TemplateDef = "dim var_Picture1"
oGauge.TemplateDef = var_Picture1
oGauge.Template = "var_Picture1.Name = `Guage_Needle.png`"

oGauge.EndUpdate()

```

## Visual Objects

```

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"
oDCOCX_Exontrol1:Layers:Add("background"):Background:Picture:Name :=
"Guage_Background.png"
oDCOCX_Exontrol1:Layers:Add("needle"):Background:Picture:Name :=
"Guage_Needle.png"
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge

oGauge = ole_1.Object
oGauge.BeginUpdate()

```

```
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"  
oGauge.Layers.Add("background").Background.Picture.Name =  
"Guage_Background.png"  
oGauge.Layers.Add("needle").Background.Picture.Name = "Guage_Needle.png"  
oGauge.EndUpdate()
```

## Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Send ComBeginUpdate  
    Set ComPicturesPath to "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"  
    Variant voLayers  
    Get ComLayers to voLayers  
    Handle hoLayers  
    Get Create (RefClass(cComLayers)) to hoLayers  
    Set pvComObject of hoLayers to voLayers  
        Variant voLayer  
        Get ComAdd of hoLayers "background" to voLayer  
        Handle hoLayer  
        Get Create (RefClass(cComLayer)) to hoLayer  
        Set pvComObject of hoLayer to voLayer  
            Variant voBackground  
            Get ComBackground of hoLayer to voBackground  
            Handle hoBackground  
            Get Create (RefClass(cComBackground)) to hoBackground  
            Set pvComObject of hoBackground to voBackground  
                Variant voPicture  
                Get ComPicture of hoBackground to voPicture  
                Handle hoPicture  
                Get Create (RefClass(cComPicture)) to hoPicture  
                Set pvComObject of hoPicture to voPicture  
                    Set ComName of hoPicture to "Guage_Background.png"  
                Send Destroy to hoPicture
```

```

    Send Destroy to hoBackground
    Send Destroy to hoLayer
Send Destroy to hoLayers
Variant voLayers1
Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer1
    Get ComAdd of hoLayers1 "needle" to voLayer1
    Handle hoLayer1
    Get Create (RefClass(cComLayer)) to hoLayer1
    Set pvComObject of hoLayer1 to voLayer1
        Variant voBackground1
        Get ComBackground of hoLayer1 to voBackground1
        Handle hoBackground1
        Get Create (RefClass(cComBackground)) to hoBackground1
        Set pvComObject of hoBackground1 to voBackground1
            Variant voPicture1
            Get ComPicture of hoBackground1 to voPicture1
            Handle hoPicture1
            Get Create (RefClass(cComPicture)) to hoPicture1
            Set pvComObject of hoPicture1 to voPicture1
                Set ComName of hoPicture1 to "Guage_Needle.png"
            Send Destroy to hoPicture1
        Send Destroy to hoBackground1
    Send Destroy to hoLayer1
Send Destroy to hoLayers1
Send ComEndUpdate
End_Procedure

```

## XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main

```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oGauge

oForm := XbpDialog():new( AppDesktop() )

oForm:drawingArea:clipChildren := .T.

oForm:create( „{100,100}, {640,480}„ .F. )

oForm:close := {|| PostAppEvent( xbeP\_Quit )}

oGauge := XbpActiveXControl():new( oForm:drawingArea )

oGauge:CLSID := "Exontrol.Gauge.1" /\*{91628F12-393C-44EF-A558-83ED1790AAD3}\*/

oGauge:create(„ {10,60},{610,370} )

oGauge:BeginUpdate()

oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Guage"  
oGauge:Layers():Add("background"):Background():Picture():Name :=  
"Guage\_Background.png"

oGauge:Layers():Add("needle"):Background():Picture():Name :=  
"Guage\_Needle.png"

oGauge:EndUpdate()

oForm:Show()

DO WHILE nEvent != xbeP\_Quit

nEvent := AppEvent( @mp1, @mp2, @oXbp )

oXbp:handleEvent( nEvent, mp1, mp2 )

ENDDO

RETURN

## method Layers.Clear ()

Removes all objects in a collection.

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

The Clear removes all layers from the control. The [Count](#) property specifies the number of layers in the control. The [PicturesPath](#) Specifies the path to load the pictures from. The [PicturesName](#) property specifies the expression that indicates the name of the picture to be loaded on each layer. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index Count - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers collection.

The following properties can be used to remove layers within the control:

- [Count](#) property, adds / removes layers to / from the control. For instance, Count property on 0, removes all layers from the control.
- Clear removes all layers from the control.
- [Remove](#) method, removes a layer from the control based on its index or key.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

# property Layers.Count as Long

Specifies the number of layers.

Type	Description
Long	A Long expression that specifies the count of layers within the control.

The Count property specifies the number of layers in the control. The [PicturesPath](#) Specifies the path to load the pictures from. The [PicturesName](#) property specifies the expression that indicates the name of the picture to be loaded on each layer. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index Count - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers collection.

The following properties can be used to add / remove layers within the control:

- Count property, adds / removes layers to / from the control
- [Add](#) method, adds a new layer to the control.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The Count property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following samples use the Layers.Count property to add new layers:

## VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = "`Layer` + str(value + 1) + `.png`"
    .Layers.Count = 5
    .EndUpdate
```



End With

## VB6

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + str(value + 1) + `.png`"
    .Layers.Count = 5
    .EndUpdate
End With
```

## VB.NET

```
With Exgauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + str(value + 1) + `.png`"
    .Layers.Count = 5
    .EndUpdate()
End With
```

## VB.NET for /COM

```
With AxGauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
    .PicturesName = ""Layer` + str(value + 1) + `.png`"
    .Layers.Count = 5
    .EndUpdate()
End With
```

## C++

/\*

*Copy and paste the following directives to your header file as*

*it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control Library'*

```
#import <ExGauge.dll>
using namespace EXGAUGELib;
*/
EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
>GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2");
spGauge1->PutPicturesName(L"Layer` + str(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(5);
spGauge1->EndUpdate();
```

## C++ Builder

```
Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";
Gauge1->PicturesName = L"Layer` + str(value + 1) + `.png`;
Gauge1->Layers->Count = 5;
Gauge1->EndUpdate();
```

## C#

```
exgauge1.BeginUpdate();
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";
exgauge1.PicturesName = "Layer` + str(value + 1) + `.png";
exgauge1.Layers.Count = 5;
exgauge1.EndUpdate();
```

## JScrip/JavaScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.BeginUpdate();
    Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";
    Gauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";
    Gauge1.Layers.Count = 5;
    Gauge1.EndUpdate();
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .BeginUpdate
        .PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2"
        .PicturesName = "`Layer` + str(value + 1) + `.png`"
        .Layers.Count = 5
        .EndUpdate
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```
axGauge1.BeginUpdate();
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2";
axGauge1.PicturesName = "`Layer` + str(value + 1) + `.png`";
axGauge1.Layers.Count = 5;
axGauge1.EndUpdate();
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    ;

    super();

    exgauge1.BeginUpdate();
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 2");
    exgauge1.PicturesName("`Layer` + str(value + 1) + `.png`");
    exgauge1.Layers().Count(5);
    exgauge1.EndUpdate();
}
```

## Delphi 8 (.NET only)

```
with AxGauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\\Program Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob
2';
    PicturesName := `Layer` + str(value + 1) + `.png`;
    Layers.Count := 5;
    EndUpdate();
```

```
end
```

## Delphi (standard)

```
with Gauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2';
  PicturesName := '\Layer` + str(value + 1) + `.png`;
  Layers.Count := 5;
  EndUpdate();
end
```

## VFP

```
with thisform.Gauge1
  .BeginUpdate
  .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
2"
  .PicturesName = "\Layer` + str(value + 1) + `.png`"
  .Layers.Count = 5
  .EndUpdate
endwith
```

## dBASE Plus

```
local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"
oGauge.PicturesName = "\Layer` + str(value + 1) + `.png`"
oGauge.Layers.Count = 5
oGauge.EndUpdate()
```

## XBasic (Alpha Five)

```
Dim oGauge as P
```

```
oGauge = topparent:CONTROL_ACTIVEX1.activex  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## Visual Objects

```
oDCOCX_Exontrol1:BeginUpdate()  
oDCOCX_Exontrol1:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oDCOCX_Exontrol1:PicturesName := "`Layer` + str(value + 1) + `.png`"  
oDCOCX_Exontrol1:Layers:Count := 5  
oDCOCX_Exontrol1:EndUpdate()
```

## PowerBuilder

```
OleObject oGauge  
  
oGauge = ole_1.Object  
oGauge.BeginUpdate()  
oGauge.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"  
oGauge.PicturesName = "`Layer` + str(value + 1) + `.png`"  
oGauge.Layers.Count = 5  
oGauge.EndUpdate()
```

## Visual DataFlex

```
Procedure OnCreate
```

```

Forward Send OnCreate
Send ComBeginUpdate
Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"
Set ComPicturesName to "`Layer` + str(value + 1) + `.png`"
Variant voLayers
Get ComLayers to voLayers
Handle hoLayers
Get Create (RefClass(cComLayers)) to hoLayers
Set pvComObject of hoLayers to voLayers
    Set ComCount of hoLayers to 5
Send Destroy to hoLayers
Send ComEndUpdate
End_Procedure

```

## XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

    oGauge:BeginUpdate()
    oGauge:PicturesPath := "C:\Program

```

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 2"

oGauge:PicturesName := "`Layer` + str(value + 1) + `.png`"

oGauge:Layers():Count := 5

oGauge:EndUpdate()

oForm:Show()

DO WHILE nEvent != xbeP\_Quit

nEvent := AppEvent( @mp1, @mp2, @oXbp )

oXbp:handleEvent( nEvent, mp1, mp2 )

ENDDO

RETURN



# property Layers.Item (Key as Variant) as Layer

Returns a specific Layer of the Layers collection.

Type	Description
Key as Variant	A long expression that indicates the index of the layer to be requested, or any other value that indicates the key of the layer.
Layer	A <a href="#">Layer</a> object being requested.

The Item property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index [Count](#) - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers clection.

The following properties can be used to access Layer objects in the control:

- Item property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following properties can be used to add layers within the control:

- [Count](#) property, adds layers to the control
- [Add](#) method, adds a new layer to the control.

The following properties can be used to remove layers within the control:

- [Count](#) property, removes layers from the control. For instance, Count property on 0, removes all layers from the control.
- [Clear](#) removes all layers from the control.
- [Remove](#) method, removes a layer from the control based on its index or key.

# method Layers.Remove (Key as Variant)

Removes a specific member from the Layers collection.

Type	Description
Key as Variant	A Long expression that specifies the index of the layer to be removed, or any other value to specify the key of the layer to be removed.

The Remove method removes a layer from the control based on its index or key. The [Clear](#) removes all layers from the control. The [Count](#) property specifies the number of layers in the control. The [PicturesPath](#) Specifies the path to load the pictures from. The [PicturesName](#) property specifies the expression that indicates the name of the picture to be loaded on each layer. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index Count - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers collection.

The following properties can be used to remove layers within the control:

- [Count](#) property, adds / removes layers to / from the control. For instance, Count property on 0, removes all layers from the control.
- Clear removes all layers from the control.
- [Remove](#) method, removes a layer from the control based on its index or key.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

# property Layers.VisibleCount as Long

Specifies the number of visible layers.

Type	Description
Long	A Long expression that specifies the number of visible layers.

The VisibleCount property indicates the number of visible layers within the control. The [VisibleItem](#) property gives the visible layer based on its position. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index [Count](#) - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers clection.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- [VisibleItem](#) property, gives the visible layer based on its position. The VisibleCount property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following properties can be used to add layers within the control:

- [Count](#) property, adds layers to the control
- [Add](#) method, adds a new layer to the control.

The following properties can be used to remove layers within the control:

- [Count](#) property, removes layers from the control. For instance, Count property on 0, removes all layers from the control.
- [Clear](#) removes all layers from the control.
- [Remove](#) method, removes a layer from the control based on its index or key.

## property Layers.VisibleItem (Position as Long) as Layer

Returns the visible Layer of the Layers collection, based on its position.

Type	Description
Position as Long	A long expression that specifies the position of the visible layer. The value should be between 0 and <a href="#">VisibleCount</a> - 1
Layer	A <a href="#">Layer</a> object that specifies the visible layer object

The VisibleItem property gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control. The [Item](#) property of the Layers collection accesses a Layer giving its index or key. The [Index](#) property is read-only and zero-based, which indicates that the layer with the Index property 0, it is the first layer, while the layer with the index [Count](#) - 1, is the last layer in the control ( in z-order ). The [Background](#) object holds pictures to be shown on the layer's background. The [Foreground](#) property of the Layer access the layer's Foreground object. The **for each** statement can be used to enumerate Layer objects in the Layers clection.

The following properties can be used to access Layer objects in the control:

- [Item](#) property, gives the Layer object based on its index / key. The [Count](#) property specifies the number of layers within the control
- VisibleItem property, gives the visible layer based on its position. The [VisibleCount](#) property indicates the number of visible layers within the control. The [Visible](#) property shows or hides the layer. The [ShowLayers](#) property indicates the only layers to be shown on the control.

The following properties can be used to add layers within the control:

- [Count](#) property, adds layers to the control
- [Add](#) method, adds a new layer to the control.

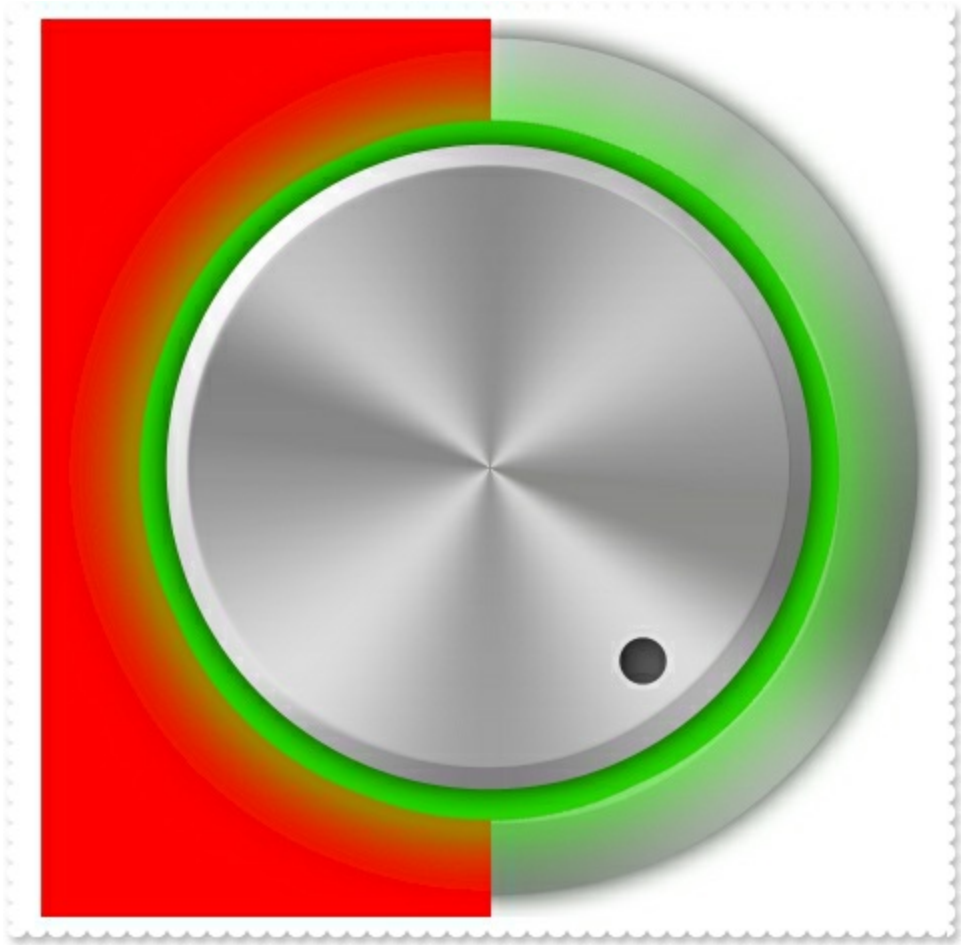
The following properties can be used to remove layers within the control:

- [Count](#) property, removes layers from the control. For instance, Count property on 0, removes all layers from the control.
- [Clear](#) removes all layers from the control.
- [Remove](#) method, removes a layer from the control based on its index or key.

# LColor object

The LColor object holds information about a solid / EBN color to be shown on the layer's background. The [Picture](#) / [ExtraPicture](#) property should be used to place a picture on the layer's background.

The following screen shot shows a layer with solid red color:



The LColor object supports the following properties and methods:

Name	Description
<a href="#">Selectable</a>	Returns or sets a value that indicates whether the color is selectable.
<a href="#">Value</a>	Indicates the solid color/visual appearance to be shown on the layer's background.
<a href="#">Visible</a>	Specifies if the color is visible or hidden.

# property LColor.Selectable as Boolean

Returns or sets a value that indicates whether the color is selectable.

Type	Description
Boolean	A Boolean expression that specifies whether the color on the layer's background is selectable.

By default, the Selectable property is True, so the user can select the layer if the cursor hovers the solid color. The Selectable property specifies whether the color on the layer's background is selectable. You can use the [Grayscale](#) property to show the entire layer in gray scale ( disable state). Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background. For instance, you can use Selectable property on False, to prevent selecting the layer when the user selects the portion of the layer that displays the solid color. The [Selectable](#) property of the Background object, affects all the pictures / colors being shown on the layer's background.

The solid / EBN color is applied on the layer's background if both of the following:

- Visible property is True
- [Value](#) property is different than -1 value

are accomplished.

# property LColor.Value as Color

Indicates the solid color/visual appearance to be shown on the layer's background.

Type	Description
Color	A Color expression that specifies the solid / EBN color to be shown on the layer's background. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used. Use the <a href="#">Add</a> method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

By default, Value property is -1, which indicates that the layer's background is transparent ( no effect ). Use the Value property to specify a solid / EBN color to be applied on the layer's background. The [Visible](#) property specifies whether the color is applied on the layer's background. The [Picture](#) / [ExtraPicture](#) property should be used to place a picture on the layer's background.

The solid / EBN color is applied on the layer's background if both of the following:

- [Visible](#) property is True
- Value property is different than -1 value

are accomplished.

The following properties can be used to move / resize the layer:

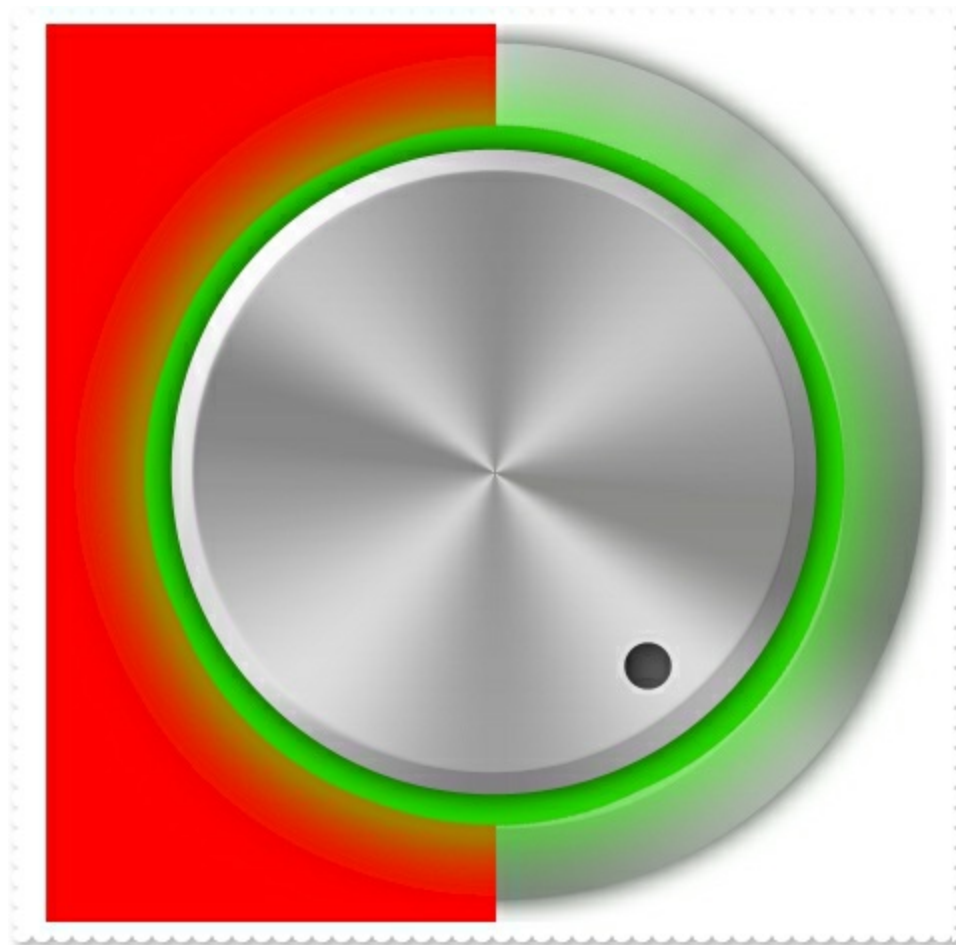
- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

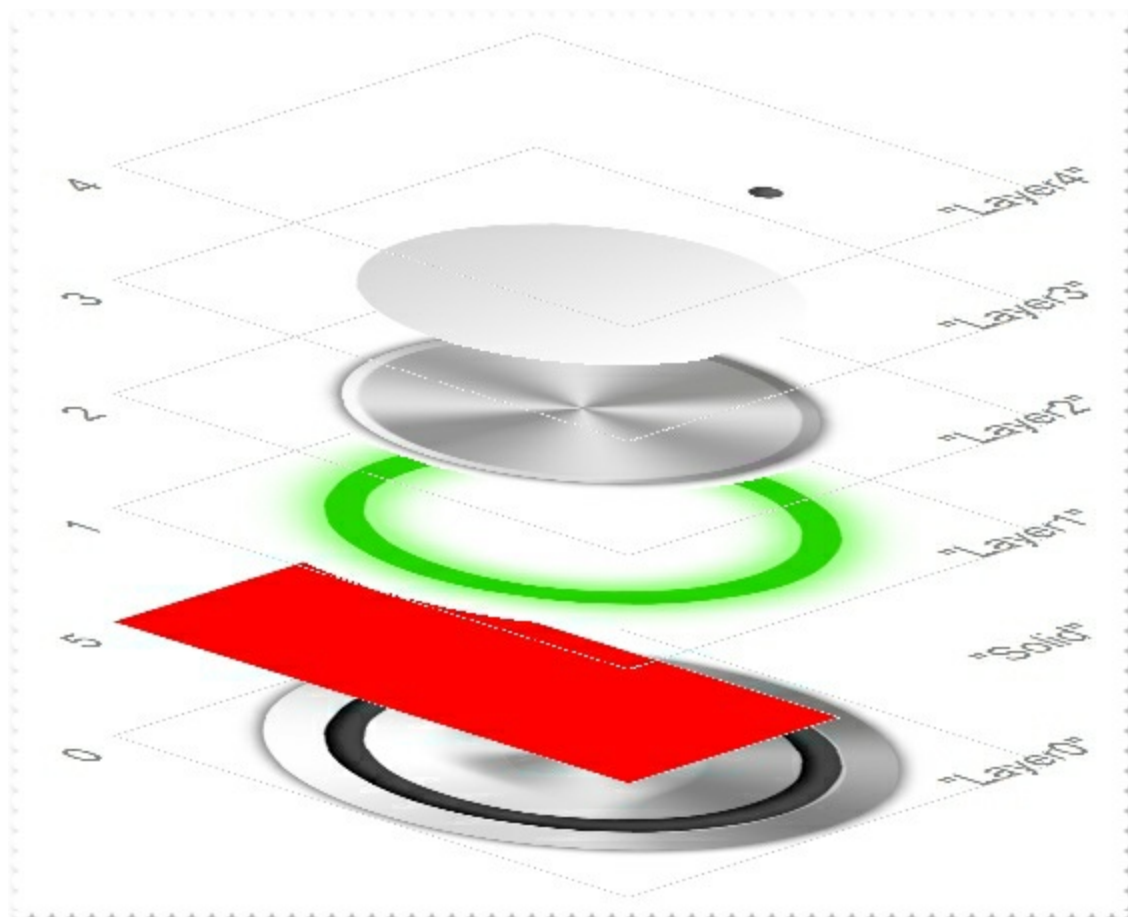
- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.



The following screen shot shows a layer with solid red color:



And if we decompose the layers we get:





The following samples show how you can apply a solid color to be display on left-half of the layer after the first visible layer:

## VBA (MS Access, Excell...)

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 5
    With .Layers.Add("Solid")
        .Position = 1
        .Width = "width/2"
        .Background.Color.Value = RGB(255,0,0)
    End With
    .EndUpdate
End With
```

## VB6

```
With Gauge1
    .BeginUpdate
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 5
    With .Layers.Add("Solid")
        .Position = 1
        .Width = "width/2"
        .Background.Color.Value = RGB(255,0,0)
    End With
    .EndUpdate
End With
```

## VB.NET

```
With Exgauge1
    .BeginUpdate()
```

```

.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = ""Layer` + int(value + 1) + `.png""
.Layers.Count = 5
With .Layers.Add("Solid")
    .Position = 1
    .Width = "width/2"
    .Background.Color.Value = Color.FromArgb(255,0,0)
End With
.EndUpdate()
End With

```

## VB.NET for /COM

```

With AxGauge1
    .BeginUpdate()
    .PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 5
    With .Layers.Add("Solid")
        .Position = 1
        .Width = "width/2"
        .Background.Color.Value = RGB(255,0,0)
    End With
    .EndUpdate()
End With

```

## C++

```

/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
    Library'

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
*/

```

```

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->BeginUpdate();
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(5);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->Add("Solid");
    var_Layer->PutPosition(1);
    var_Layer->PutWidth(L"width/2");
    var_Layer->GetBackground()->GetColor()->PutValue(RGB(255,0,0));
spGauge1->EndUpdate();

```

## C++ Builder

```

Gauge1->BeginUpdate();
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png";
Gauge1->Layers->Count = 5;
Exgauge1lib_tlb::ILayerPtr var_Layer = Gauge1->Layers->Add(TVariant("Solid"));
    var_Layer->Position = 1;
    var_Layer->Width = L"width/2";
    var_Layer->Background->Color->Value = RGB(255,0,0);
Gauge1->EndUpdate();

```

## C#

```

exgauge1.BeginUpdate();
exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
exgauge1.PicturesName = "Layer` + int(value + 1) + `.png";
exgauge1.Layers.Count = 5;
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers.Add("Solid");
    var_Layer.Position = 1;
    var_Layer.Width = "width/2";

```

```
var_Layer.Background.Color.Value = Color.FromArgb(255,0,0);  
exgauge1.EndUpdate();
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.BeginUpdate();  
    Gauge1.PicturesPath = "C:\\Program  
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 5;  
    var var_Layer = Gauge1.Layers.Add("Solid");  
        var_Layer.Position = 1;  
        var_Layer.Width = "width/2";  
        var_Layer.Background.Color.Value = 255;  
    Gauge1.EndUpdate();  
}  
</SCRIPT>  
</BODY>
```

## VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Gauge1  
        .BeginUpdate
```

```

.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 5
With .Layers.Add("Solid")
    .Position = 1
    .Width = "width/2"
    .Background.Color.Value = RGB(255,0,0)
End With
.EndUpdate
End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

axGauge1.BeginUpdate();
axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 5;
EXGAUGELib.Layer var_Layer = axGauge1.Layers.Add("Solid");
    var_Layer.Position = 1;
    var_Layer.Width = "width/2";
    var_Layer.Background.Color.Value =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
axGauge1.EndUpdate();

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Background,com_Color,com_Layer;
    anytype var_Background,var_Color,var_Layer;
    ;

```

```

super();

exgauge1.BeginUpdate();
exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob 1");
exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
exgauge1.Layers().Count(5);
var_Layer = COM::createFromObject(exgauge1.Layers()).Add("Solid"); com_Layer =
var_Layer;
    com_Layer.Position(1);
    com_Layer.Width("width/2");
    var_Background = COM::createFromObject(com_Layer.Background());
com_Background = var_Background;
    var_Color = COM::createFromObject(com_Background).Color(); com_Color =
var_Color;
    com_Color.Value(WinApi::RGB2int(255,0,0));
exgauge1.EndUpdate();
}

```

## Delphi 8 (.NET only)

```

with AxGauge1 do
begin
    BeginUpdate();
    PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
    PicturesName := '`Layer` + int(value + 1) + `.png`';
    Layers.Count := 5;
    with Layers.Add('Solid') do
    begin
        Position := 1;
        Width := 'width/2';
        Background.Color.Value := $ff;
    end;
    EndUpdate();
end

```

## Delphi (standard)

```
with Gauge1 do
begin
  BeginUpdate();
  PicturesPath := 'C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 5;
  with Layers.Add('Solid') do
  begin
    Position := 1;
    Width := 'width/2';
    Background.Color.Value := $ff;
  end;
  EndUpdate();
end
```

## VFP

```
with thisform.Gauge1
.BeginUpdate
.PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob
1"
.PicturesName = ""Layer` + int(value + 1) + `.png""
.Layers.Count = 5
with .Layers.Add("Solid")
.Position = 1
.Width = "width/2"
.Background.Color.Value = RGB(255,0,0)
endwith
.EndUpdate
endwith
```

## dBASE Plus

```
local oGauge,var_Layer
```

```

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
var_Layer = oGauge.Layers.Add("Solid")
    var_Layer.Position = 1
    var_Layer.Width = "width/2"
    var_Layer.Background.Color.Value = 0xff
oGauge.EndUpdate()

```

## XBasic (Alpha Five)

```

Dim oGauge as P
Dim var_Layer as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
var_Layer = oGauge.Layers.Add("Solid")
    var_Layer.Position = 1
    var_Layer.Width = "width/2"
    var_Layer.Background.Color.Value = 255
oGauge.EndUpdate()

```

## Visual Objects

```

local var_Layer as ILayer

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"

```



```

oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 5
var_Layer := oDCOCX_Exontrol1:Layers:Add("Solid")
    var_Layer:Position := 1
    var_Layer:Width := "width/2"
    var_Layer:Background:Color.Value := RGB(255,0,0)
oDCOCX_Exontrol1:EndUpdate()

```

## PowerBuilder

```

OleObject oGauge,var_Layer

oGauge = ole_1.Object
oGauge.BeginUpdate()
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 5
var_Layer = oGauge.Layers.Add("Solid")
    var_Layer.Position = 1
    var_Layer.Width = "width/2"
    var_Layer.Background.Color.Value = RGB(255,0,0)
oGauge.EndUpdate()

```

## Visual DataFlex

```

Procedure OnCreate
    Forward Send OnCreate
    Send ComBeginUpdate
    Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"
    Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
    Variant voLayers
    Get ComLayers to voLayers
    Handle hoLayers
    Get Create (RefClass(cComLayers)) to hoLayers

```

```
Set pvComObject of hoLayers to voLayers
  Set ComCount of hoLayers to 5
Send Destroy to hoLayers
Variant voLayers1
Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
  Variant voLayer
  Get ComAdd of hoLayers1 "Solid" to voLayer
  Handle hoLayer
  Get Create (RefClass(cComLayer)) to hoLayer
  Set pvComObject of hoLayer to voLayer
    Set ComPosition of hoLayer to 1
    Set ComWidth of hoLayer to "width/2"
    Variant voBackground
    Get ComBackground of hoLayer to voBackground
    Handle hoBackground
    Get Create (RefClass(cComBackground)) to hoBackground
    Set pvComObject of hoBackground to voBackground
      Variant voColor
      Get ComColor of hoBackground to voColor
      Handle hoColor
      Get Create (RefClass(cComColor)) to hoColor
      Set pvComObject of hoColor to voColor
        Set ComValue of hoColor to (RGB(255,0,0))
        Send Destroy to hoColor
      Send Destroy to hoBackground
    Send Destroy to hoLayer
  Send Destroy to hoLayers1
Send ComEndUpdate
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"
```

## PROCEDURE Main

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oGauge

LOCAL oLayer

oForm := XbpDialog():new( AppDesktop() )

oForm:drawingArea:clipChildren := .T.

oForm:create( ,, {100,100}, {640,480},,, .F. )

oForm:close := {|| PostAppEvent( xbeP\_Quit )}

oGauge := XbpActiveXControl():new( oForm:drawingArea )

oGauge:CLSID := "Exontrol.Gauge.1" /\*{91628F12-393C-44EF-A558-83ED1790AAD3}\*/

oGauge:create(,, {10,60},{610,370} )

oGauge:BeginUpdate()

oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob 1"

oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"

oGauge:Layers():Count := 5

oLayer := oGauge:Layers():Add("Solid")

oLayer:Position := 1

oLayer:Width := "width/2"

oLayer:Background():Color():SetProperty("Value",AutomationTranslateColor(  
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))

oGauge:EndUpdate()

oForm:Show()

DO WHILE nEvent != xbeP\_Quit

nEvent := AppEvent( @mp1, @mp2, @oXbp )

oXbp:handleEvent( nEvent, mp1, mp2 )

ENDDO

RETURN

# property LColor.Visible as Boolean

Specifies if the color is visible or hidden.

Type	Description
Boolean	A Boolean expression that specifies whether the color is applied on the layer's background.

By default, the Visible property is True, so the color is applied on the layer's background. Use the [Value](#) property to specify a solid / EBN color to be applied on the layer's background. For instance, you can use Visible property on False, to prevent showing the solid color on the layer's background. The [Selectable](#) property specifies whether the color on the layer's background is selectable. The [Visible](#) property of the Background object, affects all the pictures / colors being shown on the layer's background.

The solid / EBN color is applied on the layer's background if both of the following:

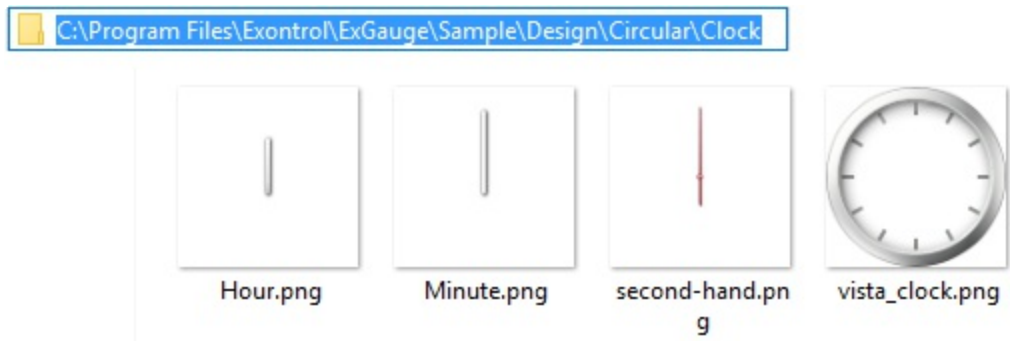
- Visible property is True
- [Value](#) property is different than -1 value

are accomplished.

# LPicture object

The LPicture object holds a picture to be displayed on the layer's background. The Layer's background can display unlimited graphics of different sizes and positions. The [Picture](#) / [ExtraPicture](#) property adds a picture on the layer's background.

For instance, having the following files in the folder C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Clock,



By loading each picture on a layer, we get something like:



The LPicture object supports the following properties and methods:

Name	Description
<a href="#">DisplayAs</a>	Retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
<a href="#">Height</a>	Specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.
<a href="#">Left</a>	Specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
<a href="#">Name</a>	Indicates the picture to be shown on the layer's background.

[Opaque](#)

Indicates if the picture is opaque or transparent.

[Selectable](#)

Returns or sets a value that indicates whether the picture is selectable.

[Top](#)

Specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.

[TransparentColorFrom](#)

Specifies the transparent color to define transparency part of the current picture (to).

[TransparentColorTo](#)

Specifies the transparent color to define transparency part of the current picture (to).

[Value](#)

Indicates the picture to be shown on the layer's background.

[Visible](#)

Specifies if the picture is shown or hidden on the layer's background.

[Width](#)

Specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.

## property LPicture.DisplayAs as PictureDisplayEnum

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

Type	Description
<a href="#">PictureDisplayEnum</a>	A PictureDisplayEnum expression that indicates the way how the graphic is displayed on the layer's background.

By default, the DisplayAs property is Stretch, that specifies that the picture is stretched on the layer's background. Use the DisplayAs property to specify a different the way how the graphic is displayed on the layer's background.

The following properties determines the position / size of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

## property LPicture.Height as String

Specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

Type	Description
String	A String value that defines the expression relative to the view/current picture, to determine the height to show the current picture on the background.

By default, the Height property is "**height**", which specifies the height in pixels of the layer where the picture is displayed. You can use the Height property of the LPicture object to show the picture with a different height. The [LayerAutoSize](#) property resizes all layers based on the picture of the first layer.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- Height, specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.



- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.

For instance, the following sample shows the picture on a size of 64,64 in the center of the layer:

```
With .Background.Picture
.Width = "64"
.Height = "64"
.Left = "(width - 64)/2"
.Top = "(height - 64)/2"
End With
```

The Height property supports the following keywords:

- **pwidth**, specifies the width in pixels of the picture object
- **pheight**, specifies the height in pixels of the picture object
- **width**, specifies the width in pixels of the layer where the picture is displayed.
- **height**, specifies the height in pixels of the layer where the picture is displayed.

The Height property supports the following constants, operators and functions:

*The constants are ( DPI-Aware components ):*

- **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 y-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** ( and 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 :*

- **MIN** ( 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.
- **MAX** ( 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  $c_1, c_2, \dots$  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  $c_1, c_2, \dots$  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 " $\%0 = c_1 ? c_1 : ( \%0 = c_2 ? c_2 : ( \dots ? \dots : default ) )$ ". 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  $\%0 \text{ switch } ('not\ found', 1, 4, 7, 9, 11)$  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 than *if* (immediate if operator) alternative.

- **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 (  $c_1, c_2, \dots$  ). For instance, if the value of expression is not any of  $c_1, c_2, \dots$  the *default\_expression* is executed and returned. If the value of the expression is  $c_1$ , 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 than using *if* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

- 0 - empty ( not initialized )
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"
- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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 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.

## property LPicture.Left as String

Specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.

Type	Description
String	A String value that defines the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.

By default, the Left property is "0". You can use the Left property of the LPicture object to show the picture moved to the left or to the right.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- Left, specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.

For instance, the following sample shows the picture on a size of 64,64 in the center of the layer:

```
With .Background.Picture
.Width = "64"
.Height = "64"
.Left = "(width - 64)/2"
.Top = "(height - 64)/2"
End With
```

The Left property supports the following keywords:

- **pwidth**, specifies the width in pixels of the picture object
- **pheight**, specifies the height in pixels of the picture object
- **width**, specifies the width in pixels of the layer where the picture is displayed.
- **height**, specifies the height in pixels of the layer where the picture is displayed.

The Left property supports the following constants, operators and functions:

*The constants are ( DPI-Aware components ):*

- **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 :*

- **MIN** ( 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.
- **MAX** ( 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 than *iif* (immediate if operator) alternative.

- ***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 than using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

- 0 - empty ( not initialized )
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"

- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startswith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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 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.

# property LPicture.Name as Variant

Indicates the picture to be shown on the layer's background.

Type	Description
Variant	<p>The Name property could be one of the following:</p> <ul style="list-style-type: none"><li>• A String expression indicates:<ul style="list-style-type: none"><li>◦ a name of a picture file in the PicturePath folder. For instance, Name = "Layer1.png", loads the Layer1.png file if found in the PicturePath folder.</li><li>◦ a picture file including its absolute path. For instance, Name = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\K loads the Layer1.png file from absolute path</li><li>◦ a key of the HTML picture, previously loaded by the HTMLPicture method. For instance, Name = "pic1", loads the HTML picture with the key pic1, so the pic1 should be load previously with a HTMLPicture call like HTMLPicture("pic1") = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\K</li><li>◦ an encode BASE64 string of a picture file. The Exontrol's <a href="#">ExImages</a> Tool encode/decode BASE64 strings from/to pictures. In this case, the string starts with "gB..", "gC.." and so on.</li></ul></li><li>• A Picture object that indicates the picture to be displayed. For instance, Name = LoadPicture("picture.jpg")</li></ul>

By default, the Name / [Value](#) property is initialized by evaluating the control's [PicturesName](#) property, whose **value** keyword is replaced by the [Index](#) of the current layer. The Name / [Value](#) property are equivalents, so use any of them do the same.

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store

bitmap digital images, independently of the display device (such as a graphics adapter)

- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

If using the **PNG** format, the control handles automatically its transparency / alpha blending, unless the [Opaque](#) property is True. For any other picture type, you can use any of the following to define the transparent region of the picture:

- [TransparentColorFrom](#), specifies the transparent color to define transparency part of the current picture (to).
- [TransparentColorTo](#), specifies the transparent color to define transparency part of the current picture (to).

The picture is displayed on the layer's background if both of the following:

- [Visible](#) property is True
- Name / [Value](#) property points to a valid picture

are accomplished.

The following properties can be used to load / import ( manually or automatically ) pictures to the layer's background:

- [PicturesPath](#) property, specifies the path to load pictures from.
- [PicturesName](#) property, specifies the expression that defines the name of the file from the PicturesPath folder to be loaded.
- Picture.Name / [Picture.Value](#) property of the Background.Picture object, defines the name of the file to be loaded ( relative, absolute, encoded or Picture object )

The [PicturesPath](#) / [PicturesName](#) properties can be used to automatically loads files from a specified folder to be displayed on the layer's background.

For instance,

PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob",  
defines default folder to load pictures from.

PicturesName = ""Layer` + str(value + 1) + `.png`", defines the name of the picture file

to be loaded by the layer with the index / value. It defines the names as: Layer1.png for the layer with the index 0, Layer2.png for the layer with the index 1, Layer3.png for the layer with the index 2, and so on.

The Picture.Name / [Picture.Value](#) property of the Picture object loads a picture / graphics to be displayed on the layer's background.

The following properties can be used to move / resize the picture on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

## property LPicture.Opaque as Boolean

Indicates if the picture is opaque or transparent.

Type	Description
Boolean	A Boolean expression that specifies whether the current picture is shown as transparent or opaque.

By default, the Opaque property is False. The Opaque property indicates if the picture is shown as opaque or transparent.

If using the **PNG** format, the control handles automatically its transparency / alpha blending, unless the Opaque property is True, so in this case, any TransparentColorFrom or TransparentColorTo property has no effect.

For any other picture type, you can use any of the following to define the transparent region of the picture:

- [TransparentColorFrom](#), specifies the transparent color to define transparency part of the current picture (to).
- [TransparentColorTo](#), specifies the transparent color to define transparency part of the current picture (to).

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

# property LPicture.Selectable as Boolean

Returns or sets a value that indicates whether the picture is selectable.

Type	Description
Boolean	A Boolean expression that specifies whether the picture on the layer's background is selectable.

By default, the Selectable property is True, so the picture is selectable on the layer's background. The Selectable property specifies whether the picture on the layer's background is selectable. Use the [Name](#) / [Value](#) property to specify a picture to be displayed on the layer's background. For instance, you can use Selectable property on False, to prevent selecting the picture on the layer's background. The [Selectable](#) property of the Background object, affects all the pictures / colors being shown on the layer's background. You can use the [Grayscale](#) property to show the entire layer in gray scale (disable state).



## property LPicture.Top as String

Specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.

Type	Description
String	A String value that defines the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.

By default, the Top property is "0". You can use the Top property of the LPicture object to show the picture moved up or down.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.
- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.

For instance, the following sample shows the picture on a size of 64,64 in the center of the layer:

```
With .Background.Picture
.Width = "64"
.Height = "64"
.Left = "(width - 64)/2"
.Top = "(height - 64)/2"
End With
```

The Top property supports the following keywords:

- **pwidth**, specifies the width in pixels of the picture object
- **pheight**, specifies the height in pixels of the picture object
- **width**, specifies the width in pixels of the layer where the picture is displayed.
- **height**, specifies the height in pixels of the layer where the picture is displayed.

The Top property supports the following constants, operators and functions:

*The constants are ( DPI-Aware components ):*

- **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 :*

- **MIN** ( 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.
- **MAX** ( 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 than *if* (immediate if operator) alternative.

- ***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 than using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

- 0 - empty ( not initialized )
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"



- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startswith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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 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.

# property LPicture.TransparentColorFrom as Color

Specifies the transparent color to define transparency part of the current picture (to).

Type	Description
Color	A Color expression that defines the transparent color.

By default, the TransparentColorFrom property is -1, which indicates that the control's [TransparentColorFrom](#) property defines the color to be shown as transparent on the current picture. The [Opaque](#) property indicates if the picture is shown as opaque or transparent.

The TransparentColorFrom / TransparentColorTo properties have effect it:

- [Opaque](#) property is False ( by default )
- picture's attribute does **not** include the PICTURE\_TRANSPARENT flag ( for instance a PNG picture with transparency, includes the PICTURE\_TRANSPARENT flag )
- TransparentColorFrom / TransparentColorTo properties points to valid colors ( different than -1 value ). For instance, if one property is defined and the other is -1, the first one defines the transparent pixels, while if both are specified and points to value different than -1, any pixel between them is considered as transparent.

If The TransparentColorFrom / TransparentColorTo properties have effect, any picture where these apply defines the pixels as:

- any pixel with a color between TransparentColorFrom and TransparentColorTo is defined as transparent
- any other pixel that's not transparent is opaque.

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics

images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

If using the **PNG** format, the control handles automatically its transparency / alpha blending ( if exists ), unless the [Opaque](#) property is True, so in this case, any TransparentColorFrom or TransparentColorTo property has no effect.

For any other picture type, you can use any of the following to define the transparent region of the picture:

- TransparentColorFrom, specifies the transparent color to define transparency part of the current picture (to).
- [TransparentColorTo](#), specifies the transparent color to define transparency part of the current picture (to).

# property LPicture.TransparentColorTo as Color

Specifies the transparent color to define transparency part of the current picture (to).

Type	Description
Color	A Color expression that defines the transparent color.

By default, the TransparentColorTo property is -1, which indicates that the control's [TransparentColorTo](#) property defines the color to be shown as transparent on the current picture. The [Opaque](#) property indicates if the picture is shown as opaque or transparent.

The TransparentColorFrom / TransparentColorTo properties have effect it:

- [Opaque](#) property is False ( by default )
- picture's attribute does **not** include the PICTURE\_TRANSPARENT flag ( for instance a PNG picture with transparency, includes the PICTURE\_TRANSPARENT flag )
- TransparentColorFrom / TransparentColorTo properties points to valid colors ( different than -1 value ). For instance, if one property is defined and the other is -1, the first one defines the transparent pixels, while if both are specified and points to value different than -1, any pixel between them is considered as transparent.

If The TransparentColorFrom / TransparentColorTo properties have effect, any picture where these apply defines the pixels as:

- any pixel with a color between TransparentColorFrom and TransparentColorTo is defined as transparent
- any other pixel that's not transparent is opaque.

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics

images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

If using the **PNG** format, the control handles automatically its transparency / alpha blending ( if saved with transparency ), unless the [Opaque](#) property is True, so in this case, any TransparentColorFrom or TransparentColorTo property has no effect.

For any other picture type, you can use any of the following to define the transparent region of the picture:

- [TransparentColorFrom](#), specifies the transparent color to define transparency part of the current picture (to).
- TransparentColorTo, specifies the transparent color to define transparency part of the current picture (to).

# property LPicture.Value as Variant

Indicates the picture to be shown on the layer's background.

Type	Description
Variant	<p>The Value property could be one of the following:</p> <ul style="list-style-type: none"><li>• A String expression indicates:<ul style="list-style-type: none"><li>◦ a name of a picture file in the PicturePath folder. For instance, Name = "Layer1.png", loads the Layer1.png file if found in the PicturePath folder.</li><li>◦ a picture file including its absolute path. For instance, Name = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\K loads the Layer1.png file from absolute path</li><li>◦ a key of the HTML picture, previously loaded by the HTMLPicture method. For instance, Name = "pic1", loads the HTML picture with the key pic1, so the pic1 should be load previously with a HTMLPicture call like HTMLPicture("pic1") = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\K</li><li>◦ an encode BASE64 string of a picture file. The Exontrol's <a href="#">ExImages</a> Tool encode/decode BASE64 strings from/to pictures. In this case, the string starts with "gB..", "gC.." and so on.</li></ul></li><li>• A Picture object that indicates the picture to be displayed. For instance, Name = LoadPicture("picture.jpg")</li></ul>

By default, the [Name](#) / Value property is initialized by evaluating the control's [PicturesName](#) property, whose **value** keyword is replaced by the [Index](#) of the current layer. The [Name](#) / Value property are equivalents, so use any of them do the same.

The control supports almost all type of pictures like

- **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store

bitmap digital images, independently of the display device (such as a graphics adapter)

- **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- **GIF** ( Graphics Interchange Format ) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.

If using the **PNG** format, the control handles automatically its transparency / alpha blending, unless the [Opaque](#) property is True. For any other picture type, you can use any of the following to define the transparent region of the picture:

- [TransparentColorFrom](#), specifies the transparent color to define transparency part of the current picture (to).
- [TransparentColorTo](#), specifies the transparent color to define transparency part of the current picture (to).

The picture is displayed on the layer's background if both of the following:

- [Visible](#) property is True
- [Name](#) / Value property points to a valid picture

are accomplished.

The following properties can be used to load / import ( manually or automatically ) pictures to the layer's background:

- [PicturesPath](#) property, specifies the path to load pictures from.
- [PicturesName](#) property, specifies the expression that defines the name of the file from the PicturesPath folder to be loaded.
- [Picture.Name](#) / Picture.Value property of the Background.Picture object, defines the name of the file to be loaded ( relative, absolute, encoded or Picture object )

The [PicturesPath](#) / [PicturesName](#) properties can be used to automatically loads files from a specified folder to be displayed on the layer's background.

For instance,

PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob",  
defines default folder to load pictures from.

PicturesName = ""Layer` + str(value + 1) + `.png`", defines the name of the picture file



to be loaded by the layer with the index / value. It defines the names as: Layer1.png for the layer with the index 0, Layer2.png for the layer with the index 1, Layer3.png for the layer with the index 2, and so on.

The [Picture.Name](#) / Picture.Value property of the Picture object loads a picture / graphics to be displayed on the layer's background.

The following properties can be used to move / resize the picture on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- [Width](#), specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

# property LPicture.Visible as Boolean

Specifies if the picture is shown or hidden on the layer's background.

Type	Description
Boolean	A Boolean expression that specifies whether the picture is visible or hidden.

By default, the Visible property is True, so the picture is visible on the layer's background. Use the [Name](#) / [Value](#) property to specify a picture to be displayed on the layer's background. For instance, you can use Visible property on False, to prevent showing the picture on the layer's background. The [Selectable](#) property specifies whether the picture on the layer's background is selectable. The [Visible](#) property of the Background object, affects all the pictures / colors being shown on the layer's background.

The picture is displayed on the layer's background if both of the following:

- Visible property is True
- [Name](#) / [Value](#) property points to a valid picture

are accomplished.

## property LPicture.Width as String

Specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.

Type	Description
String	A String value that indicates the expression relative to the view/current picture, to determine the width to show the current picture on the background.

By default, the Width property is "**width**", that specifies the width in pixels of the layer where the picture is displayed.. You can use the Width property of the LPicture object to show the picture with a different width. The [LayerAutoSize](#) property resizes all layers based on the picture of the first layer.

The following properties can be used to move / resize the **picture** on the layer's background:

- [DisplayAs](#), retrieves or sets a value that indicates the way how the graphic is displayed on the layer's background.
- [Left](#), specifies the expression relative to the view/current picture, to determine the x-position to show the current picture on the background.
- [Top](#), specifies the expression relative to the view/current picture, to determine the y-position to show the current picture on the background.
- Width, specifies the expression relative to the view/current picture, to determine the width to show the current picture on the background.
- [Height](#), specifies the expression relative to the view/current picture, to determine the height to show the current picture on the background.

The following properties determines the position / size / offset of the **layer**:

- [Left](#), specifies the expression relative to the view, to determine the x-position to show the current layer on the control.
- [Top](#), specifies the expression relative to the view, to determine the y-position to show the current layer on the control.
- [Width](#), specifies the expression relative to the view, to determine the width to show the current layer on the control.
- [Height](#), specifies the expression relative to the view, to determine the height to show the current layer on the control.

You can use the following properties to offset the view ( background + foreground ) inside the layer:

- [OffsetX](#), gets or sets a value that indicates x-offset of the layer.

- [OffsetY](#), gets or sets a value that indicates y-offset of the layer.

For instance, the following sample shows the picture on a size of 64,64 in the center of the layer:

```
With .Background.Picture
.Width = "64"
.Height = "64"
.Left = "(width - 64)/2"
.Top = "(height - 64)/2"
End With
```

The Width property supports the following keywords:

- **pwidth**, specifies the width in pixels of the picture object
- **pheight**, specifies the height in pixels of the picture object
- **width**, specifies the width in pixels of the layer where the picture is displayed.
- **height**, specifies the height in pixels of the layer where the picture is displayed.

The Width property supports the following constants, operators and functions:

*The constants are ( DPI-Aware components ):*

- **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 y-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** ( and 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 :*

- **MIN** ( 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.
- **MAX** ( 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  $c_1, c_2, \dots$  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  $c_1, c_2, \dots$  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 " $\%0 = c_1 ? c_1 : ( \%0 = c_2 ? c_2 : ( \dots ? \dots : default ) )$ ". 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  $\%0 \text{ switch } ('not\ found', 1, 4, 7, 9, 11)$  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 than *if* (immediate if operator) alternative.

- **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 (  $c_1, c_2, \dots$  ). For instance, if the value of expression is not any of  $c_1, c_2, \dots$  the *default\_expression* is executed and returned. If the value of the expression is  $c_1$ , 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  $date(shortdate(value)) \text{ case } (default:0 ; \#1/1/2002\#:1 ; \#2/1/2002\#:1 ; \#4/1/2002\#:1 ; \#5/1/2002\#:1)$  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:  $date(shortdate(value)) \text{ case } (default:0 ; \#4/1/2009\# : hour(value) \geq 6 \text{ and } hour(value) \leq 12 ; \#4/5/2009\# : hour(value) \geq 7 \text{ and } hour(value) \leq 10 \text{ or }$

*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 than using *if* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

*The supported conversion unary operators are:*

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells ( on the column 1 ) that contains string values.

Here's few predefined types:

- 0 - empty ( not initialized )
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. 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#

*Other known 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

*Other known 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"
- **startswith** (binary operator) specifies whether a string starts with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string ( 0 if not found, -1 if found ). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string ( 0 if not found, -1 if found ). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. 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 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.

# ExGauge events

The eXGauge / eXLayers library provides graphics capabilities to visually display and edit the amount, level, or contents of something. The view can show one or more layers, where each layer can display one or more transparent pictures, HTML captions which can be clipped, moved, rotated or combination of them, by dragging the mouse, rolling the mouse wheel, or using the keyboard. Using the eXGauge / eXLayers library you can easily simulate any gauges, thermometers, meters, clocks, buttons, sliders, scales, knobs, dials, switches, progress, status, indicators, LEDs, and so on.

The eXGauge supports the following events:

Name	Description
<a href="#">AnchorClick</a>	Occurs when an anchor element is clicked.
<a href="#">Change</a>	Occurs when the layer's value is changed.
<a href="#">Click</a>	Occurs when the user presses and then releases the left mouse button over the control.
<a href="#">DbClick</a>	Occurs when the user dblclk the left mouse button over an object.
<a href="#">Drag</a>	Notifies that the user drags the layer.
<a href="#">DragEnd</a>	Occurs once the user ends dragging a layer.
<a href="#">DragStart</a>	Occurs once the user starts dragging a layer.
<a href="#">Event</a>	Notifies the application once the control fires an event.
<a href="#">KeyDown</a>	Occurs when the user presses a key while an object has the focus.
<a href="#">KeyPress</a>	Occurs when the user presses and releases an ANSI key.
<a href="#">KeyUp</a>	Occurs when the user releases a key while an object has the focus.
<a href="#">MouseDown</a>	Occurs when the user presses a mouse button.
<a href="#">MouseIn</a>	Notifies that the cursor enters the layer.
<a href="#">MouseMove</a>	Occurs when the user moves the mouse.
<a href="#">MouseOut</a>	Notifies that the cursor exits the layer.
<a href="#">MouseUp</a>	Occurs when the user releases a mouse button.
<a href="#">MouseWheel</a>	Occurs when the mouse wheel moves while the control has focus
<a href="#">RClick</a>	Occurs once the user right clicks the control.



# event **AnchorClick** (AnchorID as String, Options as String)

Occurs when an anchor element is clicked.

Type	Description
AnchorID as String	A string expression that indicates the identifier of the anchor
Options as String	A string expression that specifies options of the anchor element.

The control fires the AnchorClick event to notify that the user clicks an anchor element. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The **<a>** element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The AnchorClick event is fired only if prior clicking the control it shows the hand cursor. For instance, if the cell is disabled, the hand cursor is not shown when hovers the anchor element, and so the AnchorClick event is not fired. Use the [FormatAnchor](#) property to specify the visual effect for anchor elements. For instance, if the user clicks the anchor **<a1>anchor</a>**, the control fires the AnchorClick event, where the AnchorID parameter is 1, and the Options parameter is empty. Also, if the user clicks the anchor **<a1;youreextradata>anchor</a>**, the AnchorID parameter of the AnchorClick event is 1, and the Options parameter is "youreextradata". Use the [AnchorFromPoint](#) property to retrieve the identifier of the anchor element from the cursor.

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

```
C# private void AnchorClick(object sender,string  AnchorID,string  Options)
{
}
```

```
VB Private Sub AnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles AnchorClick
End Sub
```

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

```
C# private void AnchorClick(object sender,
AxEXGAUGELib._IGaugeEvents_AnchorClickEvent e)
{
}
```

**C++**

```
void OnAnchorClick(LPCTSTR AnchorID,LPCTSTR Options)
{
}
```

**C++  
Builder**

```
void __fastcall AnchorClick(TObject *Sender,BSTR AnchorID,BSTR Options)
{
}
```

**Delphi**

```
procedure AnchorClick(ASender: TObject; AnchorID : WideString;Options :
Widestring);
begin
end;
```

**Delphi 8  
(.NET  
only)**

```
procedure AnchorClick(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_AnchorClickEvent);
begin
end;
```

**Powe...**

```
begin event AnchorClick(string AnchorID,string Options)

end event AnchorClick
```

**VB.NET**

```
Private Sub AnchorClick(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_AnchorClickEvent) Handles AnchorClick
End Sub
```

**VB6**

```
Private Sub AnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

**VBA**

```
Private Sub AnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

**VFP**

```
LPARAMETERS AnchorID,Options
```

**Xbas...**

```
PROCEDURE OnAnchorClick(oGauge,AnchorID,Options)
```



## RETURN

Syntax for AnchorClick event, **ICOM** version (others), on:

Java... `<SCRIPT EVENT="AnchorClick(AnchorID,Options)" LANGUAGE="JScript">  
</SCRIPT>`

VBSc... `<SCRIPT LANGUAGE="VBScript">  
Function AnchorClick(AnchorID,Options)  
End Function  
</SCRIPT>`

Visual  
Data... `Procedure OnComAnchorClick String llAnchorID String llOptions  
Forward Send OnComAnchorClick llAnchorID llOptions  
End_Procedure`

Visual  
Objects `METHOD OCX_AnchorClick(AnchorID,Options) CLASS MainDialog  
RETURN NIL`

X++ `void onEvent_AnchorClick(str _AnchorID,str _Options)  
{  
}`

XBasic `function AnchorClick as v (AnchorID as C,Options as C)  
end function`

dBASE `function nativeObject_AnchorClick(AnchorID,Options)  
return`

# event Change (Layer as Long)

Occurs when the layer's value is changed.

Type	Description
Layer as Long	A long expression that specifies the index of the layer whose value is being changed. The <a href="#">Item</a> property of Layers collection gets the layer based on its index.

The Change event occurs when any of the following properties:

- [Value](#), specifies the layer's value.
- [OffsetX](#), specifies a value that indicates x-offset of the layer.
- [OffsetY](#), indicates a value that indicates y-offset of the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.

are changed. For instance, you can use the Change event to update other layers when one of the layer is changed. The [OnDrag](#) property indicates the action to be performed when the user drags the layer. The [DragStart](#) / [Drag](#) / [DragEnd](#) events notify your application when a layer is dragged. The [MouseWheel](#) occurs when the mouse wheel is rolled.

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

```
C# private void Change(object sender,int Layer)
{
}
```

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

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

```
C# private void Change(object sender, AxEXGAUGELib._IGaugeEvents_ChangeEvent
e)
{
}
```

```
C++ void OnChange(long Layer)
{
}
```

```
void __fastcall Change(TObject *Sender,long  Layer)
{
}
```

Delphi

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

Delphi 8  
(.NET  
only)

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

Powe...

```
begin event Change(long  Layer)

end event Change
```

VB.NET

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

VB6

```
Private Sub Change(ByVal Layer As Long)
End Sub
```

VBA

```
Private Sub Change(ByVal Layer As Long)
End Sub
```

VFP

```
LPARAMETERS Layer
```

Xbas...

```
PROCEDURE OnChange(oGauge,Layer)

RETURN
```

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComChange Integer  IILayer  
    Forward Send OnComChange IILayer  
End_Procedure
```

Visual  
Objects

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

X++

```
void onEvent_Change(int  _Layer)  
{  
}
```

XBasic

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

dBASE

```
function nativeObject_Change(Layer)  
return
```

The following samples show how you can display the current offset, when user drags the layer:

### VBA (MS Access, Excell...)

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)  
    With Gauge1  
        .Caption(0) = .Layers.Item(Layer).OffsetX  
    End With  
End Sub
```

With Gauge1

```
.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
.PicturesName = "`Layer` + int(value + 1) + `.png`"  
.Layers.Count = 1  
With .Layers.Item(0)  
    .OnDrag = 1  
    .OffsetYValid = "0"  
End With  
End With
```

## VB6

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Gauge1_Change(ByVal Layer As Long)  
    With Gauge1  
        .Caption(exLayerCaption) = .Layers.Item(Layer).OffsetX  
    End With  
End Sub
```

With Gauge1

```
.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
.PicturesName = "`Layer` + int(value + 1) + `.png`"  
.Layers.Count = 1  
With .Layers.Item(0)  
    .OnDrag = exDoMove  
    .OffsetYValid = "0"  
End With  
End With
```

## VB.NET

*' Change event - Occurs when the layer's value is changed.*

```
Private Sub Exgauge1_Change(ByVal sender As System.Object, ByVal Layer As Integer)  
Handles Exgauge1.Change  
    With Exgauge1
```

```
.set_Caption(exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,.Layers.
```

```
End With
```

```
End Sub
```

```
With Exgauge1
```

```
.PicturesPath = "C:\Program Files
```

```
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
.PicturesName = ""Layer` + int(value + 1) + `.png`"
```

```
.Layers.Count = 1
```

```
With .Layers.Item(0)
```

```
.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoMove
```

```
.OffsetYValid = "0"
```

```
End With
```

```
End With
```

## VB.NET for /COM

```
' Change event - Occurs when the layer's value is changed.
```

```
Private Sub AxGauge1_Change(ByVal sender As System.Object, ByVal e As  
AxEXGAUGELib._IGaugeEvents_ChangeEvent) Handles AxGauge1.Change
```

```
With AxGauge1
```

```
.set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,.Layers.Item(e.lay
```

```
End With
```

```
End Sub
```

```
With AxGauge1
```

```
.PicturesPath = "C:\Program Files
```

```
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
.PicturesName = ""Layer` + int(value + 1) + `.png`"
```

```
.Layers.Count = 1
```

```
With .Layers.Item(0)
```

```
.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoMove
```

```
.OffsetYValid = "0"
```

```
End With
```

**C++**

```

// Change event - Occurs when the layer's value is changed.
void OnChangeGauge1(long Layer)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
        Library'
        #import <ExGauge.dll>
        using namespace EXGAUGELib;
    */
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    spGauge1->PutCaption(EXGAUGELib::exLayerCaption,spGauge1->GetLayers()-
> GetItem(Layer)->GetOffsetX());
}

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->PutPicturesPath(L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(1);
EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(long(0));
var_Layer->PutOnDrag(EXGAUGELib::exDoMove);
var_Layer->PutOffsetYValid(L"0");

```

**C++ Builder**

```

// Change event - Occurs when the layer's value is changed.
void __fastcall TForm1::Gauge1Change(TObject *Sender,long Layer)
{
    Gauge1->Caption[Exgaugelib_tlb::PropertyLayerCaptionEnum::exLayerCaption] =
TVariant(Gauge1->Layers->get_Item(TVariant(Layer))->OffsetX);

```

```
}
```

```
Gauge1->PicturesPath = L"C:\\Program Files  
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`";  
Gauge1->Layers->Count = 1;  
Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(0));  
var_Layer->OnDrag = Exgaugelib_tlb::OnDragLayerEnum::exDoMove;  
var_Layer->OffsetYValid = L"0";
```

## C#

```
// Change event - Occurs when the layer's value is changed.  
private void exgauge1_Change(object sender,int Layer)  
{  
  
exgauge1.set_Caption(exontrol.EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption  
  
}  
  
//this.exgauge1.Change += new  
exontrol.EXGAUGELib.exg2antt.ChangeEventHandler(this.exgauge1_Change);  
  
exgauge1.PicturesPath = "C:\\Program Files  
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
exgauge1.PicturesName = "Layer` + int(value + 1) + `.png`";  
exgauge1.Layers.Count = 1;  
exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[0];  
var_Layer.OnDrag = exontrol.EXGAUGELib.OnDragLayerEnum.exDoMove;  
var_Layer.OffsetYValid = "0";
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<SCRIPT FOR="Gauge1" EVENT="Change(Layer)" LANGUAGE="JScript">  
Gauge1.Caption(0) = Gauge1.Layers.Item(Layer).OffsetX;  
</SCRIPT>
```



```

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    Gauge1.Layers.Count = 1;
    var var_Layer = Gauge1.Layers.Item(0);
        var_Layer.OnDrag = 1;
        var_Layer.OffsetYValid = "0";
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_Change(Layer)
    With Gauge1
        .Caption(0) = .Layers.Item(Layer).OffsetX
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .PicturesPath = "C:\\Program Files

```

```
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
.PicturesName = "`Layer` + int(value + 1) + `.png`"
```

```
.Layers.Count = 1
```

```
With .Layers.Item(0)
```

```
.OnDrag = 1
```

```
.OffsetYValid = "0"
```

```
End With
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

## C# for /COM

```
// Change event - Occurs when the layer's value is changed.
```

```
private void axGauge1_Change(object sender,  
AxEXGAUGELib._IGaugeEvents_ChangeEvent e)
```

```
{
```

```
axGauge1.set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,axGau
```

```
}
```

```
//this.axGauge1.Change += new
```

```
AxEXGAUGELib._IGaugeEvents_ChangeEventHandler(this.axGauge1_Change);
```

```
axGauge1.PicturesPath = "C:\\Program Files
```

```
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
```

```
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
```

```
axGauge1.Layers.Count = 1;
```

```
EXGAUGELib.Layer var_Layer = axGauge1.Layers[0];
```

```
var_Layer.OnDrag = EXGAUGELib.OnDragLayerEnum.exDoMove;
```

```
var_Layer.OffsetYValid = "0";
```

## X++ (Dynamics Ax 2009)

```
// Change event - Occurs when the layer's value is changed.
```

```

void onEvent_Change(int _Layer)
{
    ;
    exgauge1.Caption(0/*exLayerCaption*/,exgauge1.Layers().Item(_Layer).OffsetX());
}

public void init()
{
    COM com_Layer;
    anytype var_Layer;
    ;

    super();

    exgauge1.PicturesPath("C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(1);
    var_Layer =
    COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(0));
    com_Layer = var_Layer;
    com_Layer.OnDrag(1/*exDoMove*/);
    com_Layer.OffsetYValid("0");
}

```

## Delphi 8 (.NET only)

```

// Change event - Occurs when the layer's value is changed.
procedure TWinForm1.AxGauge1_Change(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_ChangeEvent);
begin
    with AxGauge1 do
    begin

set_Caption(EXGAUGELib.PropertyLayerCaptionEnum.exLayerCaption,TObject(Layers.It

    end

```

```

end;

with AxGauge1 do
begin
  PicturesPath := 'C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';
  PicturesName := 'Layer' + int(value + 1) + '.png';
  Layers.Count := 1;
  with Layers.Item[TObject(0)] do
  begin
    OnDrag := EXGAUGELib.OnDragLayerEnum.exDoMove;
    OffsetYValid := '0';
  end;
end
end

```

## Delphi (standard)

```

// Change event - Occurs when the layer's value is changed.
procedure TForm1.Gauge1Change(ASender: TObject; Layer : Integer);
begin
  with Gauge1 do
  begin
    Caption[EXGAUGELib_TLB.exLayerCaption] :=
OleVariant(Layers.Item[OleVariant(Layer)].OffsetX);
  end
end;

with Gauge1 do
begin
  PicturesPath := 'C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';
  PicturesName := 'Layer' + int(value + 1) + '.png';
  Layers.Count := 1;
  with Layers.Item[OleVariant(0)] do
  begin
    OnDrag := EXGAUGELib_TLB.exDoMove;
    OffsetYValid := '0';
  end;
end;
end

```

```
end;  
end
```

## VFP

*\*\*\* Change event - Occurs when the layer's value is changed. \*\*\**

```
LPARAMETERS Layer
```

```
with thisform.Gauge1
```

```
.Object.Caption(0) = .Layers.Item(Layer).OffsetX
```

```
endwith
```

```
with thisform.Gauge1
```

```
.PicturesPath = "C:\Program Files
```

```
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
.PicturesName = ""Layer` + int(value + 1) + `.png`"
```

```
.Layers.Count = 1
```

```
with .Layers.Item(0)
```

```
.OnDrag = 1
```

```
.OffsetYValid = "0"
```

```
endwith
```

```
endwith
```

## dBASE Plus

```
/*
```

```
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
```

```
Change = class::nativeObject_Change
```

```
endwith
```

```
*/
```

```
// Occurs when the layer's value is changed.
```

```
function nativeObject_Change(Layer)
```

```
oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
```

```
oGauge.Template = [Caption(0) = Layers.Item(Layer).OffsetX] // oGauge.Caption(0)  
= oGauge.Layers.Item(Layer).OffsetX
```

```
return
```

```
local oGauge,var_Layer
```

```

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 1
var_Layer = oGauge.Layers.Item(0)
    var_Layer.OnDrag = 1
    var_Layer.OffsetYValid = "0"

```

## XBasic (Alpha Five)

```

' Occurs when the layer's value is changed.
function Change as v (Layer as N)
    oGauge = topparent:CONTROL_ACTIVEX1.activex
    oGauge.Template = "Caption(0) = Layers.Item(Layer).OffsetX" // oGauge.Caption(0)
    = oGauge.Layers.Item(Layer).OffsetX
end function

Dim oGauge as P
Dim var_Layer as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 1
var_Layer = oGauge.Layers.Item(0)
    var_Layer.OnDrag = 1
    var_Layer.OffsetYValid = "0"

```

## Visual Objects

```

METHOD OCX_Exontrol1Change(Layer) CLASS MainDialog
    // Change event - Occurs when the layer's value is changed.
    oDCOCX_Exontrol1:[Caption,exLayerCaption] := oDCOCX_Exontrol1:Layers:
    [Item,Layer]:OffsetX

```

RETURN NIL

local var\_Layer as ILayer

oDCOCX\_Exontrol1:PicturesPath := "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"

oDCOCX\_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"

oDCOCX\_Exontrol1:Layers.Count := 1

var\_Layer := oDCOCX\_Exontrol1:Layers:[Item,0]

var\_Layer:OnDrag := exDoMove

var\_Layer:OffsetYValid := "0"

## PowerBuilder

```
/*begin event Change(long Layer) - Occurs when the layer's value is changed.*/  
/*  
    oGauge = ole_1.Object  
    oGauge.Caption(0,oGauge.Layers.Item(Layer).OffsetX)  
*/  
/*end event Change*/
```

OleObject oGauge,var\_Layer

oGauge = ole\_1.Object

oGauge.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"

oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"

oGauge.Layers.Count = 1

var\_Layer = oGauge.Layers.Item(0)

var\_Layer.OnDrag = 1

var\_Layer.OffsetYValid = "0"

## Visual DataFlex

```
// Occurs when the layer's value is changed.
```

```
Procedure OnComChange Integer llLayer
```

```
Forward Send OnComChange IILayer
Variant v
Variant voLayers
Get ComLayers to voLayers
Handle hoLayers
Get Create (RefClass(cComLayers)) to hoLayers
Set pvComObject of hoLayers to voLayers
    Variant voLayer
    Get ComItem of hoLayers IILayer to voLayer
    Handle hoLayer
    Get Create (RefClass(cComLayer)) to hoLayer
    Set pvComObject of hoLayer to voLayer
        Get ComOffsetX of hoLayer to v
    Send Destroy to hoLayer
Send Destroy to hoLayers
Set ComCaption OLEexLayerCaption to v
End_Procedure
```

#### Procedure OnCreate

```
Forward Send OnCreate
Set ComPicturesPath to "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers1
Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Set ComCount of hoLayers1 to 1
Send Destroy to hoLayers1
Variant voLayers2
Get ComLayers to voLayers2
Handle hoLayers2
Get Create (RefClass(cComLayers)) to hoLayers2
Set pvComObject of hoLayers2 to voLayers2
    Variant voLayer1
    Get ComItem of hoLayers2 0 to voLayer1
```



```

Handle hoLayer1
Get Create (RefClass(cComLayer)) to hoLayer1
Set pvComObject of hoLayer1 to voLayer1
    Set ComOnDrag of hoLayer1 to OLEexDoMove
    Set ComOffsetYValid of hoLayer1 to "0"
Send Destroy to hoLayer1
Send Destroy to hoLayers2
End_Procedure

```

## XBase++

```

PROCEDURE OnChange(oGauge,Layer)

```

```

oGauge:SetProperty("Caption",0/*exLayerCaption*/,oGauge:Layers:Item(Layer):OffsetX)

```

```

RETURN

```

```

#include "AppEvent.ch"

```

```

#include "ActiveX.ch"

```

```

PROCEDURE Main

```

```

    LOCAL oForm

```

```

    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

```

```

    LOCAL oGauge

```

```

    LOCAL oLayer

```

```

    oForm := XbpDialog():new( AppDesktop() )

```

```

    oForm:drawingArea:clipChildren := .T.

```

```

    oForm:create( ,, {100,100}, {640,480},,, .F. )

```

```

    oForm:close := {|| PostAppEvent( xbpQuit )}

```

```

    oGauge := XbpActiveXControl():new( oForm:drawingArea )

```

```

    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/

```

```

    oGauge:create(,, {10,60},{610,370} )

```

```

    oGauge:Change := {|| Layer| OnChange(oGauge,Layer)} /*Occurs when the layer's

```

*value is changed.\*/\**

```
oGauge:PicturesPath := "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
oGauge:Layers():Count := 1
oLayer := oGauge:Layers:Item(0)
oLayer:OnDrag := 1/*exDoMove*/
oLayer:OffsetYValid := "0"

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

# event Click ()

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

## Type

## Description

The Click event is fired when the user releases the left mouse button over the control. Use a [MouseDown](#) or [MouseUp](#) event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The Click event is not fired if you click, drag and release the mouse over the control. The [OnDrag](#) property indicates the action to be performed when the user clicks and drags the layer.

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(oGauge)  
  
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComClick
    Forward Send OnComClick
End_Procedure
```

Visual  
Objects

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

X++

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

XBasic

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

dBASE

```
function nativeObject_Click()
return
```

# event DbtClick (Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS)

Occurs when the user dbtclk the left mouse button over an object.

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

The DbtClick event is fired when user double clicks the control. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The Click event is not fired if you click, drag and release the mouse over the control. The [OnDrag](#) property indicates the action to be performed when the user clicks and drags the layer.

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

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

VBPrivate Sub DbtClick(ByVal sender As System.Object,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles DbtClick  
End Sub

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

C#private void DbtClick(object sender, AxEXGAUGELib.\_IGaugeEvents\_DbtClickEvent e)  
{  
}

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

C++  
Builder

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

Delphi

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

Delphi 8  
(.NET  
only)

```
procedure DbClick(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_DblClickEvent);
begin
end;
```

Powe...

```
begin event DbClick(integer  Shift,long  X,long  Y)

end event DbClick
```

VB.NET

```
Private Sub DbClick(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_DblClickEvent) Handles DbClick
End Sub
```

VB6

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

VBA

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

VFP

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

Xbas...

```
PROCEDURE OnDbClick(oGauge,Shift,X,Y)

RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComDbClick Short  IIShift OLE_XPOS_PIXELS  IIX OLE_YPOS_PIXELS  
IYY  
    Forward Send OnComDbClick IIShift IIX IYY  
End_Procedure
```

Visual  
Objects

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

X++

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

XBasic

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

dBASE

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



# event Drag (DragInfo as DragInfo)

Notifies that the user drags the layer.

Type	Description
DragInfo as <a href="#">DragInfo</a>	A DragInfo object that carries information about the dragging operation.

Any layer on the control supports drag operations like moving, rotation, or combination of them, when the user clicks and drags a layer. The drag operation automatically starts when the user clicks a visible, selectable and draggable layer. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). The [Visible](#) property shows or hides a specific layer (visible). The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [Change](#) event occurs when the layer's value is changed.

The control fires the drag events in the following order:

- [DragStart](#) event notifies that a layer begins to drag. You can use the DragStart event to cancel the dragging operation.
- Drag event notifies that the layer is dragging. You can use the Drag event to perform other actions, on any layer during the dragging operation.
- [DragEnd](#) event notifies that the dragging the layer ends. You can use the DragEnd event to perform other actions, on any layer when dragging operation ends.

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

```
C# private void Drag(object sender,exontrol.EXGAUGELib.DragInfo  DragInfo)
{
}
```

```
VB Private Sub Drag(ByVal sender As System.Object,ByVal DragInfo As
exontrol.EXGAUGELib.DragInfo) Handles Drag
End Sub
```

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

```
C# private void Drag(object sender, AxEXGAUGELib._IGaugeEvents_DragEvent e)
{
}
```

```
C++ void OnDrag(LPDISPATCH  DragInfo)
```

```
{  
}
```

C++  
Builder

```
void __fastcall Drag(TObject *Sender,Exgaugelib_tlb::IDragInfo  *DragInfo)  
{  
}
```

Delphi

```
procedure Drag(ASender: TObject; DragInfo : IDragInfo);  
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure Drag(sender: System.Object; e:  
AxEXGAUGELib._IGaugeEvents_DragEvent);  
begin  
end;
```

Powe...

```
begin event Drag(oleobject  DragInfo)  
  
end event Drag
```

VB.NET

```
Private Sub Drag(ByVal sender As System.Object, ByVal e As  
AxEXGAUGELib._IGaugeEvents_DragEvent) Handles Drag  
End Sub
```

VB6

```
Private Sub Drag(ByVal DragInfo As EXGAUGELibCtl.IDragInfo)  
End Sub
```

VBA

```
Private Sub Drag(ByVal DragInfo As Object)  
End Sub
```

VFP

```
LPARAMETERS DragInfo
```

Xbas...

```
PROCEDURE OnDrag(oGauge,DragInfo)  
  
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComDrag Variant IIDragInfo  
    Forward Send OnComDrag IIDragInfo  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_Drag(DragInfo) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Drag(COM _DragInfo)  
{  
}
```

XBasic

```
function Drag as v (DragInfo as OLE::Exontrol.Gauge.1::IIDragInfo)  
end function
```

dBASE

```
function nativeObject_Drag(DragInfo)  
return
```

# event DragEnd (DragInfo as DragInfo, Cancel as Boolean)

Occurs once the user ends dragging a layer.

Type	Description
DragInfo as <a href="#">DragInfo</a>	A DragInfo object that carries information about the dragging operation.
Cancel as Boolean	A Boolean expression that specifies whether the dragging operation was canceled, for instance, the user presses the ESC during dragging operation.

Any layer on the control supports drag operations like moving, rotation, or combination of them, when the user clicks and drags a layer. The drag operation automatically starts when the user clicks a visible, selectable and draggable layer. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). The [Visible](#) property shows or hides a specific layer (visible). The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [Change](#) event occurs when the layer's value is changed.

The control fires the drag events in the following order:

- [DragStart](#) event notifies that a layer begins to drag. You can use the DragStart event to cancel the dragging operation.
- [Drag](#) event notifies that the layer is dragging. You can use the Drag event to perform other actions, on any layer during the dragging operation.
- DragEnd event notifies that the dragging the layer ends. You can use the DragEnd event to perform other actions, on any layer when dragging operation ends.

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

```
C# private void DragEnd(object sender,exontrol.EXGAUGELib.DragInfo
DragInfo,bool  Cancel)
{
}
```

```
VB Private Sub DragEnd(ByVal sender As System.Object,ByVal DragInfo As
exontrol.EXGAUGELib.DragInfo,ByVal Cancel As Boolean) Handles DragEnd
End Sub
```

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

```
C# private void DragEnd(object sender,
```

```
AxEXGAUGELib._IGaugeEvents_DragEndEvent e)
{
}
```

```
C++ void OnDragEnd(LPDISPATCH DragInfo,BOOL Cancel)
{
}
```

```
C++ Builder void __fastcall DragEnd(TObject *Sender,Exgaugelib_tlb::IDragInfo
*DragInfo,VARIANT_BOOL Cancel)
{
}
```

```
Delphi procedure DragEnd(ASender: TObject; DragInfo : IDragInfo;Cancel : WordBool);
begin
end;
```

```
Delphi 8 (.NET only) procedure DragEnd(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_DragEndEvent);
begin
end;
```

```
Powe... begin event DragEnd(oleobject DragInfo,boolean Cancel)

end event DragEnd
```

```
VB.NET Private Sub DragEnd(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_DragEndEvent) Handles DragEnd
End Sub
```

```
VB6 Private Sub DragEnd(ByVal DragInfo As EXGAUGELibCtl.IDragInfo,ByVal Cancel As
Boolean)
End Sub
```

```
VBA Private Sub DragEnd(ByVal DragInfo As Object,ByVal Cancel As Boolean)
End Sub
```

```
VFP LPARAMETERS DragInfo,Cancel
```

Xbas...

```
PROCEDURE OnDragEnd(oGauge,DragInfo,Cancel)

RETURN
```

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

Java...

```
<SCRIPT EVENT="DragEnd(DragInfo,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function DragEnd(DragInfo,Cancel)
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComDragEnd Variant  IIDragInfo Boolean  IICancel
    Forward Send OnComDragEnd IIDragInfo IICancel
End_Procedure
```

Visual  
Objects

```
METHOD OCX_DragEnd(DragInfo,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_DragEnd(COM  _DragInfo,boolean  _Cancel)
{
}
```

XBasic

```
function DragEnd as v (DragInfo as OLE::Exontrol.Gauge.1::IIDragInfo,Cancel as L)
end function
```

dBASE

```
function nativeObject_DragEnd(DragInfo,Cancel)
return
```

# event DragStart (DragInfo as DragInfo, ByRef Cancel as Boolean)

Occurs once the user starts dragging a layer.

Type	Description
DragInfo as <a href="#">DragInfo</a>	A DragInfo object that carries information about the dragging operation. You can use <a href="#">UserData</a> property of the DragInfo object to associate any-extra data to the current dragging operation.
Cancel as Boolean	(By Reference) A Boolean expression that specifies whether the dragging operation should be canceled, or can continue. By default, the Cancel parameter is False, so you can change the Cancel parameter during the DragStart event to prevent dragging any layer on the control.

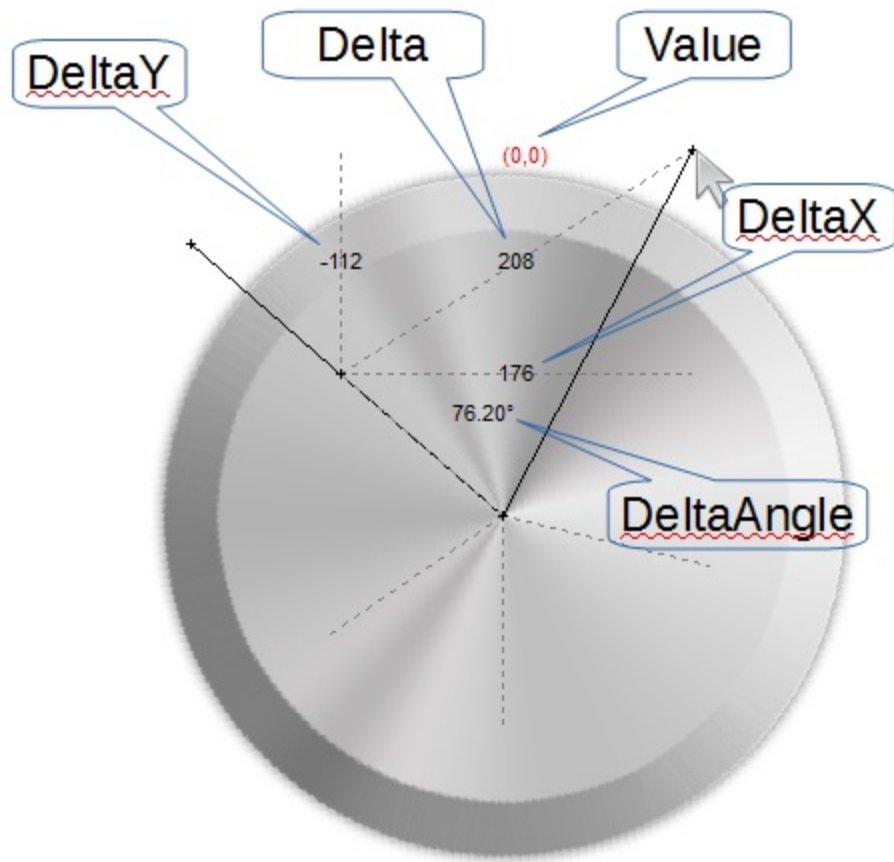
Any layer on the control supports drag operations like moving, rotation, or combination of them, when the user clicks and drags a layer. The drag operation automatically starts when the user clicks a visible, selectable and draggable layer. The [OnDrag](#) property indicates the action to be performed when the user drags the layer ( draggable ). The [Visible](#) property shows or hides a specific layer (visible). The [Selectable](#) property returns or sets a value that indicates whether the layer is selectable. The [Change](#) event occurs when the layer's value is changed.

The control fires the drag events in the following order:

- DragStart event notifies that a layer begins to drag. You can use the DragStart event to cancel the dragging operation.
- [Drag](#) event notifies that the layer is dragging. You can use the Drag event to perform other actions, on any layer during the dragging operation.
- [DragEnd](#) event notifies that the dragging the layer ends. You can use the DragEnd event to perform other actions, on any layer when dragging operation ends.

You can use the [Debug](#) property of the DragInfo object to display debugging information during dragging.

The following screen show shows debugging information during dragging:



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

```
C# private void DragStart(object sender,exontrol.EXGAUGELib.DragInfo  DragInfo,ref
bool  Cancel)
{
}
```

```
VB Private Sub DragStart(ByVal sender As System.Object,ByVal DragInfo As
exontrol.EXGAUGELib.DragInfo,ByRef Cancel As Boolean) Handles DragStart
End Sub
```

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

```
C# private void DragStart(object sender,
AxEXGAUGELib._IGaugeEvents_DragStartEvent e)
{
}
```

```
C++ void OnDragStart(LPDISPATCH  DragInfo,BOOL FAR*  Cancel)
{
}
```



**C++ Builder**

```
void __fastcall DragStart(TObject *Sender,Exgaugelib_tlb::IDragInfo
*DragInfo,VARIANT_BOOL *  Cancel)
{
}
```

**Delphi**

```
procedure DragStart(ASender: TObject; DragInfo : IDragInfo;var Cancel :
WordBool);
begin
end;
```

**Delphi 8  
(.NET  
only)**

```
procedure DragStart(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_DragStartEvent);
begin
end;
```

**Powe...**

```
begin event DragStart(oleobject  DragInfo,boolean  Cancel)

end event DragStart
```

**VB.NET**

```
Private Sub DragStart(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_DragStartEvent) Handles DragStart
End Sub
```

**VB6**

```
Private Sub DragStart(ByVal DragInfo As EXGAUGELibCtl.IDragInfo,Cancel As
Boolean)
End Sub
```

**VBA**

```
Private Sub DragStart(ByVal DragInfo As Object,Cancel As Boolean)
End Sub
```

**VFP**

```
LPARAMETERS DragInfo,Cancel
```

**Xbas...**

```
PROCEDURE OnDragStart(oGauge,DragInfo,Cancel)

RETURN
```

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

Java... `<SCRIPT EVENT="DragStart(DragInfo,Cancel)" LANGUAGE="JScript">  
</SCRIPT>`

VBSc... `<SCRIPT LANGUAGE="VBScript">  
Function DragStart(DragInfo,Cancel)  
End Function  
</SCRIPT>`

Visual  
Data... `Procedure OnComDragStart Variant IIDragInfo Boolean IICancel  
Forward Send OnComDragStart IIDragInfo IICancel  
End_Procedure`

Visual  
Objects `METHOD OCX_DragStart(DragInfo,Cancel) CLASS MainDialog  
RETURN NIL`

X++ `void onEvent_DragStart(COM _DragInfo,COMVariant /*bool*/ _Cancel)  
{  
}`

XBasic `function DragStart as v (DragInfo as OLE::Exontrol.Gauge.1::IIDragInfo,Cancel as  
L)  
end function`

dBASE `function nativeObject_DragStart(DragInfo,Cancel)  
return`

# 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 <a href="#">EventParam(-2)</a> 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.

Here's how the output is shown, when printing the EventParam(-2) during the Event event:

```
MouseIn/2( 0 )
MouseMove/-606( 0 , 0 , 184 , 420 )
MouseDown/-605( 1 , 0 , 184 , 420 )
DragStart/4( [Object] , =false )
MouseMove/-606( 1 , 0 , 185 , 420 )
Change/7( 0 )
Drag/5( [Object] )
DragEnd/6( [Object] , false )
MouseUp/-607( 1 , 0 , 337 , 382 )
MouseMove/-606( 0 , 0 , 338 , 383 )
MouseOut/3( 0 )
MouseMove/-606( 0 , 0 , 369 , 623 )
MouseMove/-606( 0 , 0 , 369 , 636 )
```

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, AxEXGAUGELib._IGaugeEvents_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:
AxEXGAUGELib._IGaugeEvents_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
AxEXGAUGELib._IGaugeEvents_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(oGauge,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 (ByRef KeyCode as Integer, Shift as Integer)

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

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

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

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

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

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

C#

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

VB

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

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

C#

```
private void KeyDownEvent(object sender,
AxEXGAUGELib._IGaugeEvents_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:  
AxEXGAUGELib._IGaugeEvents_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  
AxEXGAUGELib._IGaugeEvents_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(oGauge,KeyCode,Shift)
```



RETURN

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

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

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

Visual  
Data... Procedure OnComKeyDown Short llKeyCode Short llShift  
Forward Send OnComKeyDown llKeyCode llShift  
End\_Procedure

Visual  
Objects METHOD OCX\_KeyDown(KeyCode,Shift) CLASS MainDialog  
RETURN NIL

X++ void onEvent\_KeyDown(COMVariant /\*short\*/ \_KeyCode,int \_Shift)  
{  
}

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

dBASE function nativeObject\_KeyDown(KeyCode,Shift)  
return

# event KeyPress (ByRef KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

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

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

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

C#

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

VB

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

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

C#

```
private void KeyPressEvent(object sender,
AxEXGAUGELib._IGaugeEvents_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:  
AxEXGAUGELib.\_IGaugeEvents\_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  
AxEXGAUGELib.\_IGaugeEvents\_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(oGauge,KeyAscii)  
  
RETURN

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

**Java...** <SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function KeyPress(KeyAscii)  
End Function

```
</SCRIPT>
```

Visual  
Data...

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

Visual  
Objects

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

X++

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

XBasic

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

dBASE

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

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

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

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

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

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

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

VBPrivate Sub KeyUp(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal Shift As Short) Handles KeyUpEnd Sub

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

C#private void KeyUpEvent(object sender,AxEXGAUGELib.\_IGaugeEvents\_KeyUpEvent e){}

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

C++ Buildervoid \_\_fastcall KeyUp(TObject \*Sender,short \* KeyCode,short Shift){}

```
}
```

Delphi

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

Delphi 8  
(.NET  
only)

```
procedure KeyUpEvent(sender: System.Object; e:  
AxEXGAUGELib._IGaugeEvents_KeyUpEvent);  
begin  
end;
```

Power...

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

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As  
AxEXGAUGELib._IGaugeEvents_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(oGauge,KeyCode,Shift)  
  
RETURN
```

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

Java...

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

VBSc...

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

```
Function KeyUp(KeyCode,Shift)
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComKeyUp Short  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 [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The Click event is not fired if you click, drag and release the mouse over the control. The [OnDrag](#) property indicates the action to be performed when the user clicks and drags the layer.

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,  
AxEXGAUGELib._IGaugeEvents_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:  
AxEXGAUGELib._IGaugeEvents_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  
AxEXGAUGELib._IGaugeEvents_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(oGauge,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
IIX OLE_YPOS_PIXELS Ily
    Forward Send OnComMouseDown llButton llShift IIX Ily
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.Gauge.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Gauge.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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



# event MouseIn (Layer as Long)

Notifies that the cursor enters the layer.

Type	Description
Layer as Long	A long expression that specifies the index of the layer where the cursor is entering. The <a href="#">Item</a> property of Layers collection gets the layer based on its index.

The MouseIn / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects. The [AllowSmoothChange](#) property specifies the properties of the layers that support smooth change. For instance, you can use the MouseIn / MouseOut event to change gradually the brightness / contrast or the transparency, of the layer, while the [AllowSmoothChange](#) property is not exSmoothChangeless.

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

C#private void MouseIn(object sender,int Layer){}

VBPrivate Sub MouseIn(ByVal sender As System.Object,ByVal Layer As Integer)Handles MouseInEnd Sub

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

C#private void MouseIn(object sender, AxEXGAUGELib.\_IGaugeEvents\_MouseInEvent e){}

C++void OnMouseIn(long Layer){}

C++ Buildervoid \_\_fastcall MouseIn(TObject \*Sender,long Layer){}

Delphi

```
procedure MouseIn(ASender: TObject; Layer : Integer);  
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure MouseIn(sender: System.Object; e:  
AxEXGAUGELib._IGaugeEvents_MouseInEvent);  
begin  
end;
```

Power...

```
begin event MouseIn(long Layer)  
  
end event MouseIn
```

VB.NET

```
Private Sub MouseIn(ByVal sender As System.Object, ByVal e As  
AxEXGAUGELib._IGaugeEvents_MouseInEvent) Handles MouseIn  
End Sub
```

VB6

```
Private Sub MouseIn(ByVal Layer As Long)  
End Sub
```

VBA

```
Private Sub MouseIn(ByVal Layer As Long)  
End Sub
```

VFP

```
LPARAMETERS Layer
```

Xbas...

```
PROCEDURE OnMouseIn(oGauge,Layer)  
  
RETURN
```

Syntax for MouseIn event, **ICOM** version (others), on:

Java...

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

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function MouseIn(Layer)
```

```
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComMouseIn Integer  IILayer
    Forward Send OnComMouseIn IILayer
End_Procedure
```

Visual  
Objects

```
METHOD OCX_MouseIn(Layer) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_MouseIn(int  _Layer)
{
}
```

XBasic

```
function MouseIn as v (Layer  as N)
end function
```

dBASE

```
function nativeObject_MouseIn(Layer)
return
```

The following samples shows how you can change the layer's brightness when the cursor enters / leaves the layer:

### VBA (MS Access, Excell...)

*' MouseIn event - Notifies that the cursor enters the layer.*

```
Private Sub Gauge1_MouseIn(ByVal Layer As Long)
    With Gauge1
        With .Layers.Item(Layer)
            .Brightness(1) = 100
            .Brightness(2) = 0
            .Brightness(3) = 0
        End With
    End With
End Sub
```

*' MouseOut event - Notifies that the cursor exits the layer.*

```
Private Sub Gauge1_MouseOut(ByVal Layer As Long)
```

```

With Gauge1
    With .Layers.Item(Layer)
        .Brightness(1) = Gauge1.DefaultLayer(128)
        .Brightness(2) = Gauge1.DefaultLayer(128)
        .Brightness(3) = Gauge1.DefaultLayer(128)
    End With
End With
End Sub

With Gauge1
    .DefaultLayer(128) = 51
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 1
End With

```

## VB6

```

' MouseIn event - Notifies that the cursor enters the layer.
Private Sub Gauge1_MouseIn(ByVal Layer As Long)
    With Gauge1
        With .Layers.Item(Layer)
            .Brightness(exRedChannel) = 100
            .Brightness(exGreenChannel) = 0
            .Brightness(exBlueChannel) = 0
        End With
    End With
End Sub

' MouseOut event - Notifies that the cursor exits the layer.
Private Sub Gauge1_MouseOut(ByVal Layer As Long)
    With Gauge1
        With .Layers.Item(Layer)
            .Brightness(exRedChannel) = Gauge1.DefaultLayer(exDefLayerBrightness)
            .Brightness(exGreenChannel) = Gauge1.DefaultLayer(exDefLayerBrightness)
            .Brightness(exBlueChannel) = Gauge1.DefaultLayer(exDefLayerBrightness)
        End With
    End With
End Sub

```

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

```
With Gauge1
```

```
    .DefaultLayer(exDefLayerBrightness) = 51
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 1
End With
```

## VB.NET

```
' MouseIn event - Notifies that the cursor enters the layer.
```

```
Private Sub Exgauge1_MouseIn(ByVal sender As System.Object, ByVal Layer As Integer)
```

```
Handles Exgauge1.MouseIn
```

```
    With Exgauge1
```

```
        With .Layers.Item(Layer)
```

```
        .set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel, 100)
```

```
        .set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel, 100)
```

```
        .set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel, 100)
```

```
    End With
```

```
End With
```

```
End Sub
```

```
' MouseOut event - Notifies that the cursor exits the layer.
```

```
Private Sub Exgauge1_MouseOut(ByVal sender As System.Object, ByVal Layer As Integer) Handles Exgauge1.MouseOut
```

```
    With Exgauge1
```

```
        With .Layers.Item(Layer)
```



```

.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel,Ex

.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel

.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel,Ex

    End With
    End With
End Sub

With Exgauge1

.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightne

    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 1
End With

```

## VB.NET for /COM

```

' MouseIn event - Notifies that the cursor enters the layer.
Private Sub AxGauge1_MouseIn(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_MouseInEvent) Handles AxGauge1.MouseIn
    With AxGauge1
        With .Layers.Item(e.layer)
            .Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel) =
100
            .Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel) = 0
            .Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel) = 0
        End With
    End With
End Sub

```

*' MouseOut event - Notifies that the cursor exits the layer.*

```
Private Sub AxGauge1_MouseOut(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_MouseOutEvent) Handles AxGauge1.MouseOut
    With AxGauge1
        With .Layers.Item(e.layer)
            Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel) =
AxGauge1.DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness

            Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel) =
AxGauge1.DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness

            Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel) =
AxGauge1.DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness

        End With
    End With
End Sub

With AxGauge1

.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness,51)
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 1
End With
```

**C++**

*// MouseIn event - Notifies that the cursor enters the layer.*

```
void OnMouseInGauge1(long Layer)
```

```
{
```

```
    /*
```

*Copy and paste the following directives to your header file as  
it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control  
Library'*

```

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
    */
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(Layer);
    var_Layer->PutBrightness(EXGAUGELib::exRedChannel,100);
    var_Layer->PutBrightness(EXGAUGELib::exGreenChannel,0);
    var_Layer->PutBrightness(EXGAUGELib::exBlueChannel,0);
}

// MouseOut event - Notifies that the cursor exits the layer.
void OnMouseOutGauge1(long Layer)
{
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(Layer);
    var_Layer->PutBrightness(EXGAUGELib::exRedChannel,spGauge1-
> GetDefaultLayer(EXGAUGELib::exDefLayerBrightness));
    var_Layer->PutBrightness(EXGAUGELib::exGreenChannel,spGauge1-
> GetDefaultLayer(EXGAUGELib::exDefLayerBrightness));
    var_Layer->PutBrightness(EXGAUGELib::exBlueChannel,spGauge1-
> GetDefaultLayer(EXGAUGELib::exDefLayerBrightness));
}

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerBrightness,long(51));
spGauge1->PutPicturesPath(L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(1);

```

## C++ Builder

```

// MouseIn event - Notifies that the cursor enters the layer.

```

```

void __fastcall TForm1::Gauge1MouseIn(TObject *Sender,long  Layer)
{
    Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(Layer));
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exRedChannel,100);
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exGreenChannel,0);
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exBlueChannel,0);
}

```

*// MouseOut event - Notifies that the cursor exits the layer.*

```

void __fastcall TForm1::Gauge1MouseOut(TObject *Sender,long  Layer)
{
    Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(Layer));
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exRedChannel,Gauge1-
> get_DefaultLayer(Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness));

    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exGreenChannel,Gauge1-
> get_DefaultLayer(Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness));

    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exBlueChannel,Gauge1-
> get_DefaultLayer(Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness));

}

```

```

Gauge1-
> DefaultLayer[Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness] =
TVariant(51);
Gauge1->PicturesPath = L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`;
Gauge1->Layers->Count = 1;

```

*// MouseIn event - Notifies that the cursor enters the layer.*

```
private void exgauge1_MouseIn(object sender,int Layer)
{
    exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[Layer];

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedC

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGree

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueC

}
//this.exgauge1.MouseIn += new
exontrol.EXGAUGELib.exg2antt.MouseInEventHandler(this.exgauge1_MouseIn);
```

*// MouseOut event - Notifies that the cursor exits the layer.*

```
private void exgauge1_MouseOut(object sender,int Layer)
{
    exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[Layer];

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedC

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGree

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueC

}
//this.exgauge1.MouseOut += new
exontrol.EXGAUGELib.exg2antt.MouseOutEventHandler(this.exgauge1_MouseOut);

exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayC
```

```
exgauge1.PicturesPath = "C:\\Program Files  
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
exgauge1.Layers.Count = 1;
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<SCRIPT FOR="Gauge1" EVENT="MouseIn(Layer)" LANGUAGE="JScript">  
    var var_Layer = Gauge1.Layers.Item(Layer);  
    var_Layer.Brightness(1) = 100;  
    var_Layer.Brightness(2) = 0;  
    var_Layer.Brightness(3) = 0;  
</SCRIPT>  
  
<SCRIPT FOR="Gauge1" EVENT="MouseOut(Layer)" LANGUAGE="JScript">  
    var var_Layer = Gauge1.Layers.Item(Layer);  
    var_Layer.Brightness(1) = Gauge1.DefaultLayer(128);  
    var_Layer.Brightness(2) = Gauge1.DefaultLayer(128);  
    var_Layer.Brightness(3) = Gauge1.DefaultLayer(128);  
</SCRIPT>  
  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.DefaultLayer(128) = 51;  
    Gauge1.PicturesPath = "C:\\Program Files  
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 1;  
}  
</SCRIPT>
```

</BODY>

## VBScript

```
<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_MouseIn(Layer)
  With Gauge1
    With .Layers.Item(Layer)
      .Brightness(1) = 100
      .Brightness(2) = 0
      .Brightness(3) = 0
    End With
  End With
End Function
</SCRIPT>
```

```
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_MouseOut(Layer)
  With Gauge1
    With .Layers.Item(Layer)
      .Brightness(1) = Gauge1.DefaultLayer(128)
      .Brightness(2) = Gauge1.DefaultLayer(128)
      .Brightness(3) = Gauge1.DefaultLayer(128)
    End With
  End With
End Function
</SCRIPT>
```

```
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Gauge1
    .DefaultLayer(128) = 51
```

```

        .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = "`Layer` + int(value + 1) + `.png`"
        .Layers.Count = 1
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

// MouseIn event - Notifies that the cursor enters the layer.
private void axGauge1_MouseIn(object sender,
AxEXGAUGELib._IGaugeEvents_MouseInEvent e)
{
    EXGAUGELib.Layer var_Layer = axGauge1.Layers[e.layer];

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel,1

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChanne

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel,C

}
//this.axGauge1.MouseIn += new
AxEXGAUGELib._IGaugeEvents_MouseInEventHandler(this.axGauge1_MouseIn);

// MouseOut event - Notifies that the cursor exits the layer.
private void axGauge1_MouseOut(object sender,
AxEXGAUGELib._IGaugeEvents_MouseOutEvent e)
{
    EXGAUGELib.Layer var_Layer = axGauge1.Layers[e.layer];

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel,a

```



```

var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel, a
var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel, a
}
//this.axGauge1.MouseOut += new
AxEXGAUGELib._IGaugeEvents_MouseOutEventHandler(this.axGauge1_MouseOut);

axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightn

axGauge1.PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "\\Layer` + int(value + 1) + `.png`;
axGauge1.Layers.Count = 1;

```

## X++ (Dynamics Ax 2009)

```

// MouseIn event - Notifies that the cursor enters the layer.
void onEvent_MouseIn(int _Layer)
{
    COM com_Layer;
    anytype var_Layer;
    ;
    var_Layer = COM::createFromObject(exgauge1.Layers()).Item(_Layer); com_Layer =
var_Layer;
    com_Layer.Brightness(1/*exRedChannel*/,100);
    com_Layer.Brightness(2/*exGreenChannel*/,0);
    com_Layer.Brightness(3/*exBlueChannel*/,0);
}

// MouseOut event - Notifies that the cursor exits the layer.
void onEvent_MouseOut(int _Layer)
{

```

```

COM com_Layer;
anytype var_Layer;
;
var_Layer = COM::createFromObject(exgauge1.Layers()).Item(_Layer); com_Layer =
var_Layer;

com_Layer.Brightness(1/*exRedChannel*/,exgauge1.DefaultLayer(128/*exDefLayerBrig

com_Layer.Brightness(2/*exGreenChannel*/,exgauge1.DefaultLayer(128/*exDefLayerB

com_Layer.Brightness(3/*exBlueChannel*/,exgauge1.DefaultLayer(128/*exDefLayerBrig

}

public void init()
{
    COM com_Layer;
    anytype var_Layer;
    ;

    super();

exgauge1.DefaultLayer(128/*exDefLayerBrightness*/,COMVariant::createFromInt(51));
    exgauge1.PicturesPath("C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(1);
}

```

## Delphi 8 (.NET only)

```

// MouseIn event - Notifies that the cursor enters the layer.
procedure TForm1.AxGauge1_MouseIn(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_MouseInEvent);

```

```

begin
  with AxGauge1 do
    begin
      with Layers.Item[TObject(e.layer)] do
        begin
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel] :=
100;
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel] :=
0;
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel] := 0;
        end;
      end
    end;
end;

// MouseOut event - Notifies that the cursor exits the layer.
procedure TWinForm1.AxGauge1_MouseOut(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_MouseOutEvent);
begin
  with AxGauge1 do
    begin
      with Layers.Item[TObject(e.layer)] do
        begin
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel] :=
AxGauge1.DefaultLayer[EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness];

          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel] :=
AxGauge1.DefaultLayer[EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness];

          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel] :=
AxGauge1.DefaultLayer[EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness];

        end;
      end
    end;
end;

with AxGauge1 do
  begin

```

```
set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness,TObject
```

```
    PicturesPath := 'C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';  
    PicturesName := 'Layer` + int(value + 1) + `.png`';  
    Layers.Count := 1;  
end
```

## Delphi (standard)

```
// MouseIn event - Notifies that the cursor enters the layer.
```

```
procedure TForm1.Gauge1MouseIn(ASender: TObject; Layer : Integer);  
begin  
    with Gauge1 do  
    begin  
        with Layers.Item[OleVariant(Layer)] do  
        begin  
            Brightness[EXGAUGELib_TLB.exRedChannel] := 100;  
            Brightness[EXGAUGELib_TLB.exGreenChannel] := 0;  
            Brightness[EXGAUGELib_TLB.exBlueChannel] := 0;  
        end;  
    end  
end;
```

```
// MouseOut event - Notifies that the cursor exits the layer.
```

```
procedure TForm1.Gauge1MouseOut(ASender: TObject; Layer : Integer);  
begin  
    with Gauge1 do  
    begin  
        with Layers.Item[OleVariant(Layer)] do  
        begin  
            Brightness[EXGAUGELib_TLB.exRedChannel] :=  
Gauge1.DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness];  
            Brightness[EXGAUGELib_TLB.exGreenChannel] :=  
Gauge1.DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness];  
            Brightness[EXGAUGELib_TLB.exBlueChannel] :=
```

```

Gauge1.DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness];
    end;
end
end;

with Gauge1 do
begin
    DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness] := OleVariant(51);
    PicturesPath := 'C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 1;
end

```

## VFP

*\*\*\* MouseIn event - Notifies that the cursor enters the layer. \*\*\**

```

LPARAMETERS Layer
with thisform.Gauge1
    with .Layers.Item(Layer)
        .Brightness(1) = 100
        .Brightness(2) = 0
        .Brightness(3) = 0
    endwith
endwith

```

*\*\*\* MouseOut event - Notifies that the cursor exits the layer. \*\*\**

```

LPARAMETERS Layer
with thisform.Gauge1
    with .Layers.Item(Layer)
        .Brightness(1) = thisform.Gauge1.DefaultLayer(128)
        .Brightness(2) = thisform.Gauge1.DefaultLayer(128)
        .Brightness(3) = thisform.Gauge1.DefaultLayer(128)
    endwith
endwith

```

```

with thisform.Gauge1

```

```

.Object.DefaultLayer(128) = 51
.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "`Layer` + int(value + 1) + `.png`"
.Layers.Count = 1
endwith

```

## dBASE Plus

```

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    MouseIn = class::nativeObject_MouseIn
endwith
*/
// Notifies that the cursor enters the layer.
function nativeObject_MouseIn(Layer)
    local var_Layer
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    var_Layer = oGauge.Layers.Item(Layer)
    // var_Layer.Brightness(1) = 100
    with (oGauge)
        TemplateDef = [dim var_Layer]
        TemplateDef = var_Layer
        Template = [var_Layer.Brightness(1) = 100]
    endwith
    // var_Layer.Brightness(2) = 0
    with (oGauge)
        TemplateDef = [dim var_Layer]
        TemplateDef = var_Layer
        Template = [var_Layer.Brightness(2) = 0]
    endwith
    // var_Layer.Brightness(3) = 0
    with (oGauge)
        TemplateDef = [dim var_Layer]
        TemplateDef = var_Layer
        Template = [var_Layer.Brightness(3) = 0]
    endwith

```

return

*/\*  
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)*

*MouseOut = class::nativeObject\_MouseOut*

*endwith*

*\*/*

*// Notifies that the cursor exits the layer.*

function nativeObject\_MouseOut(Layer)

local var\_Layer

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject

var\_Layer = oGauge.Layers.Item(Layer)

*// var\_Layer.Brightness(1) = oGauge.DefaultLayer(128)*

with (oGauge)

TemplateDef = [dim var\_Layer]

TemplateDef = var\_Layer

Template = [var\_Layer.Brightness(1) = Me.DefaultLayer(128)]

endwith

*// var\_Layer.Brightness(2) = oGauge.DefaultLayer(128)*

with (oGauge)

TemplateDef = [dim var\_Layer]

TemplateDef = var\_Layer

Template = [var\_Layer.Brightness(2) = Me.DefaultLayer(128)]

endwith

*// var\_Layer.Brightness(3) = oGauge.DefaultLayer(128)*

with (oGauge)

TemplateDef = [dim var\_Layer]

TemplateDef = var\_Layer

Template = [var\_Layer.Brightness(3) = Me.DefaultLayer(128)]

endwith

return

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject

oGauge.Template = [DefaultLayer(128) = 51] *// oGauge.DefaultLayer(128) = 51*

oGauge.PicturesPath = "C:\Program Files

```
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 1
```

## XBasic (Alpha Five)

*' Notifies that the cursor enters the layer.*

```
function MouseIn as v (Layer as N)  
    Dim var_Layer as P  
    oGauge = topparent:CONTROL_ACTIVEX1.activex  
    var_Layer = oGauge.Layers.Item(Layer)  
    ' var_Layer.Brightness(1) = 100  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(1) = 100"  
    ' var_Layer.Brightness(2) = 0  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(2) = 0"  
    ' var_Layer.Brightness(3) = 0  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(3) = 0"
```

end function

*' Notifies that the cursor exits the layer.*

```
function MouseOut as v (Layer as N)  
    Dim var_Layer as P  
    oGauge = topparent:CONTROL_ACTIVEX1.activex  
    var_Layer = oGauge.Layers.Item(Layer)  
    ' var_Layer.Brightness(1) = oGauge.DefaultLayer(128)  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(1) = Me.DefaultLayer(128)"  
    ' var_Layer.Brightness(2) = oGauge.DefaultLayer(128)
```



```

oGauge.TemplateDef = "dim var_Layer"
oGauge.TemplateDef = var_Layer
oGauge.Template = "var_Layer.Brightness(2) = Me.DefaultLayer(128)"
' var_Layer.Brightness(3) = oGauge.DefaultLayer(128)
oGauge.TemplateDef = "dim var_Layer"
oGauge.TemplateDef = var_Layer
oGauge.Template = "var_Layer.Brightness(3) = Me.DefaultLayer(128)"

end function

Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.Template = "DefaultLayer(128) = 51" // oGauge.DefaultLayer(128) = 51
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = ""Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 1

```

## Visual Objects

```

METHOD OCX_Exontrol1MouseIn(Layer) CLASS MainDialog
// MouseIn event - Notifies that the cursor enters the layer.
local var_Layer as ILayer
var_Layer := oDCOCX_Exontrol1:Layers:[Item,Layer]
var_Layer:[Brightness,exRedChannel] := 100
var_Layer:[Brightness,exGreenChannel] := 0
var_Layer:[Brightness,exBlueChannel] := 0
RETURN NIL

METHOD OCX_Exontrol1MouseOut(Layer) CLASS MainDialog
// MouseOut event - Notifies that the cursor exits the layer.
local var_Layer as ILayer
var_Layer := oDCOCX_Exontrol1:Layers:[Item,Layer]
var_Layer:[Brightness,exRedChannel] := oDCOCX_Exontrol1:
[DefaultLayer,exDefLayerBrightness]

```

```

    var_Layer:[Brightness,exGreenChannel] := oDCOCX_Exontrol1:
[DefaultLayer,exDefLayerBrightness]
    var_Layer:[Brightness,exBlueChannel] := oDCOCX_Exontrol1:
[DefaultLayer,exDefLayerBrightness]
RETURN NIL

oDCOCX_Exontrol1:[DefaultLayer,exDefLayerBrightness] := 51
oDCOCX_Exontrol1:PicturesPath := "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 1

```

## PowerBuilder

```

/*begin event MouseIn(long Layer) - Notifies that the cursor enters the layer.*/
/*
    OleObject var_Layer
    oGauge = ole_1.Object
    var_Layer = oGauge.Layers.Item(Layer)
        var_Layer.Brightness(1,100)
        var_Layer.Brightness(2,0)
        var_Layer.Brightness(3,0)
*/
/*end event MouseIn*/

/*begin event MouseOut(long Layer) - Notifies that the cursor exits the layer.*/
/*
    OleObject var_Layer
    oGauge = ole_1.Object
    var_Layer = oGauge.Layers.Item(Layer)
        var_Layer.Brightness(1,oGauge.DefaultLayer(128))
        var_Layer.Brightness(2,oGauge.DefaultLayer(128))
        var_Layer.Brightness(3,oGauge.DefaultLayer(128))
*/
/*end event MouseOut*/

```

OleObject oGauge

oGauge = ole\_1.Object

oGauge.DefaultLayer(128,51)

oGauge.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"

oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"

oGauge.Layers.Count = 1

## Visual DataFlex

*// Notifies that the cursor enters the layer.*

Procedure OnComMouseIn Integer IILayer

Forward Send OnComMouseIn IILayer

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComItem of hoLayers IILayer to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Set **ComBrightness** of hoLayer OLEexRedChannel to 100

Set **ComBrightness** of hoLayer OLEexGreenChannel to 0

Set **ComBrightness** of hoLayer OLEexBlueChannel to 0

Send Destroy to hoLayer

Send Destroy to hoLayers

End\_Procedure

*// Notifies that the cursor exits the layer.*

Procedure OnComMouseOut Integer IILayer

Forward Send OnComMouseOut IILayer

Variant voLayers1

```

Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer1
        Get ComItem of hoLayers1 llLayer to voLayer1
        Handle hoLayer1
        Get Create (RefClass(cComLayer)) to hoLayer1
        Set pvComObject of hoLayer1 to voLayer1
            Variant v
                Get ComDefaultLayer OLEexDefLayerBrightness to v
                Set ComBrightness of hoLayer1 OLEexRedChannel to v
            Variant v1
                Get ComDefaultLayer OLEexDefLayerBrightness to v1
                Set ComBrightness of hoLayer1 OLEexGreenChannel to v1
            Variant v2
                Get ComDefaultLayer OLEexDefLayerBrightness to v2
                Set ComBrightness of hoLayer1 OLEexBlueChannel to v2
        Send Destroy to hoLayer1
    Send Destroy to hoLayers1
End_Procedure

```

#### Procedure OnCreate

```

Forward Send OnCreate
Set ComDefaultLayer OLEexDefLayerBrightness to 51
Set ComPicturesPath to "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers2
Get ComLayers to voLayers2
Handle hoLayers2
Get Create (RefClass(cComLayers)) to hoLayers2
Set pvComObject of hoLayers2 to voLayers2
    Set ComCount of hoLayers2 to 1
    Send Destroy to hoLayers2
End_Procedure

```

```
PROCEDURE OnMouseIn(oGauge,Layer)
```

```
    LOCAL oLayer
```

```
    oLayer := oGauge:Layers:Item(Layer)
```

```
        oLayer:SetProperty("Brightness",1/*exRedChannel*/,100)
```

```
        oLayer:SetProperty("Brightness",2/*exGreenChannel*/,0)
```

```
        oLayer:SetProperty("Brightness",3/*exBlueChannel*/,0)
```

```
RETURN
```

```
PROCEDURE OnMouseOut(oGauge,Layer)
```

```
    LOCAL oLayer
```

```
    oLayer := oGauge:Layers:Item(Layer)
```

```
oLayer:SetProperty("Brightness",1/*exRedChannel*/,oGauge:DefaultLayer(128/*exDefL
```

```
oLayer:SetProperty("Brightness",2/*exGreenChannel*/,oGauge:DefaultLayer(128/*exDe
```

```
oLayer:SetProperty("Brightness",3/*exBlueChannel*/,oGauge:DefaultLayer(128/*exDefl
```

```
RETURN
```

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oGauge
```

```
oForm := XbpDialog():new( AppDesktop() )
```

```
oForm:drawingArea:clipChildren := .T.
```

```
oForm:create( „{100,100}, {640,480},„ .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

    oGauge:MouseIn := {|Layer| OnMouseIn(oGauge,Layer)} /*Notifies that the
cursor enters the layer.*/
    oGauge:MouseOut := {|Layer| OnMouseOut(oGauge,Layer)} /*Notifies that the
cursor exits the layer.*/

    oGauge:SetProperty("DefaultLayer",128/*exDefLayerBrightness*/,51)
    oGauge:PicturesPath := "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 1

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

# event MouseEventArgs (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. Gets which mouse button was pressed as 1 for Left Mouse Button, 2 for Right Mouse Button and 4 for Middle Mouse Button.
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 MouseEventArgs event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseEventArgs event whenever the mouse position is within its borders. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The Click event is not fired if you click, drag and release the mouse over the control. The [OnDrag](#) property indicates the action to be performed when the user clicks and drags the layer. The [MouseIn](#) / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer.

Syntax for MouseEventArgs event, /NET version, on:

C#

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

VB

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

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

**C#**

```
private void MouseMoveEvent(object sender,
AxEXGAUGELib._IGaugeEvents_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:
AxEXGAUGELib._IGaugeEvents_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
AxEXGAUGELib._IGaugeEvents_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(oGauge,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
IIX OLE_YPOS_PIXELS  ILY
    Forward Send OnComMouseMove llButton llShift IIX ILY
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.Gauge.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Gauge.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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



# event MouseOut (Layer as Long)

Notifies that the cursor exits the layer.

Type	Description
Layer as Long	A long expression that specifies the index of the layer where the cursor is leaving. The <a href="#">Item</a> property of Layers collection gets the layer based on its index.

The [MouseIn](#) / MouseOut event notifies your application when the cursor is entering / leaving the layer. The [MouseMove](#) event is generated continually as the mouse pointer moves across objects. The [AllowSmoothChange](#) property specifies the properties of the layers that support smooth change. For instance, you can use the MouseIn / MouseOut event to change gradually the brightness / contrast or the transparency, of the layer, while the [AllowSmoothChange](#) property is not exSmoothChangeless.

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

C#private void MouseOut(object sender,int Layer){}

VBPrivate Sub MouseOut(ByVal sender As System.Object,ByVal Layer As Integer)Handles MouseOutEnd Sub

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

C#private void MouseOut(object sender,AxEXGAUGELib.\_IGaugeEvents\_MouseOutEvent e){}

C++void OnMouseOut(long Layer){}

C++ Buildervoid \_\_fastcall MouseOut(TObject \*Sender,long Layer){}

Delphi

```
procedure MouseOut(ASender: TObject; Layer : Integer);  
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure MouseOut(sender: System.Object; e:  
AxEXGAUGELib._IGaugeEvents_MouseOutEvent);  
begin  
end;
```

Power...

```
begin event MouseOut(long Layer)  
  
end event MouseOut
```

VB.NET

```
Private Sub MouseOut(ByVal sender As System.Object, ByVal e As  
AxEXGAUGELib._IGaugeEvents_MouseOutEvent) Handles MouseOut  
End Sub
```

VB6

```
Private Sub MouseOut(ByVal Layer As Long)  
End Sub
```

VBA

```
Private Sub MouseOut(ByVal Layer As Long)  
End Sub
```

VFP

```
LPARAMETERS Layer
```

Xbas...

```
PROCEDURE OnMouseOut(oGauge,Layer)  
  
RETURN
```

Syntax for MouseOut event, **ICOM** version (others), on:

Java...

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

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function MouseOut(Layer)
```

```
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComMouseOut Integer ILayer
    Forward Send OnComMouseOut ILayer
End_Procedure
```

Visual  
Objects

```
METHOD OCX_MouseOut(Layer) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_MouseOut(int _Layer)
{
}
```

XBasic

```
function MouseOut as v (Layer as N)
end function
```

dBASE

```
function nativeObject_MouseOut(Layer)
return
```

The following samples shows how you can change the layer's brightness when the cursor enters / leaves the layer:

### VBA (MS Access, Excell...)

*' MouseIn event - Notifies that the cursor enters the layer.*

```
Private Sub Gauge1_MouseIn(ByVal Layer As Long)
    With Gauge1
        With .Layers.Item(Layer)
            .Brightness(1) = 100
            .Brightness(2) = 0
            .Brightness(3) = 0
        End With
    End With
End Sub
```

*' MouseOut event - Notifies that the cursor exits the layer.*

```
Private Sub Gauge1_MouseOut(ByVal Layer As Long)
```

```

With Gauge1
    With .Layers.Item(Layer)
        .Brightness(1) = Gauge1.DefaultLayer(128)
        .Brightness(2) = Gauge1.DefaultLayer(128)
        .Brightness(3) = Gauge1.DefaultLayer(128)
    End With
End With
End Sub

With Gauge1
    .DefaultLayer(128) = 51
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 1
End With

```

## VB6

*' MouseIn event - Notifies that the cursor enters the layer.*

```

Private Sub Gauge1_MouseIn(ByVal Layer As Long)
    With Gauge1
        With .Layers.Item(Layer)
            .Brightness(exRedChannel) = 100
            .Brightness(exGreenChannel) = 0
            .Brightness(exBlueChannel) = 0
        End With
    End With
End Sub

```

*' MouseOut event - Notifies that the cursor exits the layer.*

```

Private Sub Gauge1_MouseOut(ByVal Layer As Long)
    With Gauge1
        With .Layers.Item(Layer)
            .Brightness(exRedChannel) = Gauge1.DefaultLayer(exDefLayerBrightness)
            .Brightness(exGreenChannel) = Gauge1.DefaultLayer(exDefLayerBrightness)
            .Brightness(exBlueChannel) = Gauge1.DefaultLayer(exDefLayerBrightness)
        End With
    End With
End Sub

```

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

```
With Gauge1
```

```
    .DefaultLayer(exDefLayerBrightness) = 51
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 1
End With
```

## VB.NET

```
' MouseIn event - Notifies that the cursor enters the layer.
```

```
Private Sub Exgauge1_MouseIn(ByVal sender As System.Object, ByVal Layer As Integer)
```

```
Handles Exgauge1.MouseIn
```

```
    With Exgauge1
```

```
        With .Layers.Item(Layer)
```

```
        .set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel, 100)
```

```
        .set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel, 100)
```

```
        .set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel, 100)
```

```
    End With
```

```
End With
```

```
End Sub
```

```
' MouseOut event - Notifies that the cursor exits the layer.
```

```
Private Sub Exgauge1_MouseOut(ByVal sender As System.Object, ByVal Layer As Integer) Handles Exgauge1.MouseOut
```

```
    With Exgauge1
```

```
        With .Layers.Item(Layer)
```

```

.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel,Ex

.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel

.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel,Ex

    End With
    End With
End Sub

With Exgauge1

.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightne

    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 1
End With

```

## VB.NET for /COM

```

' MouseIn event - Notifies that the cursor enters the layer.
Private Sub AxGauge1_MouseIn(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_MouseInEvent) Handles AxGauge1.MouseIn
    With AxGauge1
        With .Layers.Item(e.layer)
            .Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel) =
100
            .Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel) = 0
            .Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel) = 0
        End With
    End With
End Sub

```



*' MouseOut event - Notifies that the cursor exits the layer.*

```
Private Sub AxGauge1_MouseOut(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_MouseOutEvent) Handles AxGauge1.MouseOut
    With AxGauge1
        With .Layers.Item(e.layer)
            Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel) =
AxGauge1.DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness

            Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel) =
AxGauge1.DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness

            Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel) =
AxGauge1.DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness

        End With
    End With
End Sub

With AxGauge1

.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness,51)
    .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = ""Layer` + int(value + 1) + `.png""
    .Layers.Count = 1
End With
```

**C++**

*// MouseIn event - Notifies that the cursor enters the layer.*

```
void OnMouseInGauge1(long Layer)
```

```
{
```

```
    /*
```

*Copy and paste the following directives to your header file as  
it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control  
Library'*

```

    #import <ExGauge.dll>
    using namespace EXGAUGELib;
    */
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(Layer);
    var_Layer->PutBrightness(EXGAUGELib::exRedChannel,100);
    var_Layer->PutBrightness(EXGAUGELib::exGreenChannel,0);
    var_Layer->PutBrightness(EXGAUGELib::exBlueChannel,0);
}

// MouseOut event - Notifies that the cursor exits the layer.
void OnMouseOutGauge1(long Layer)
{
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()->GetItem(Layer);
    var_Layer->PutBrightness(EXGAUGELib::exRedChannel,spGauge1-
> GetDefaultLayer(EXGAUGELib::exDefLayerBrightness));
    var_Layer->PutBrightness(EXGAUGELib::exGreenChannel,spGauge1-
> GetDefaultLayer(EXGAUGELib::exDefLayerBrightness));
    var_Layer->PutBrightness(EXGAUGELib::exBlueChannel,spGauge1-
> GetDefaultLayer(EXGAUGELib::exDefLayerBrightness));
}

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerBrightness,long(51));
spGauge1->PutPicturesPath(L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(1);

```

## C++ Builder

```

// MouseIn event - Notifies that the cursor enters the layer.

```

```

void __fastcall TForm1::Gauge1MouseIn(TObject *Sender,long  Layer)
{
    Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(Layer));
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exRedChannel,100);
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exGreenChannel,0);
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exBlueChannel,0);
}

```

*// MouseOut event - Notifies that the cursor exits the layer.*

```

void __fastcall TForm1::Gauge1MouseOut(TObject *Sender,long  Layer)
{
    Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers->get_Item(TVariant(Layer));
    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exRedChannel,Gauge
> get_DefaultLayer(Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness));

    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exGreenChannel,Gauge
> get_DefaultLayer(Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness));

    var_Layer-
> set_Brightness(Exgaugelib_tlb::ColorAdjustmentChannelEnum::exBlueChannel,Gauge
> get_DefaultLayer(Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness));

}

```

```

Gauge1-
> DefaultLayer[Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerBrightness] =
TVariant(51);
Gauge1->PicturesPath = L"C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png`;
Gauge1->Layers->Count = 1;

```

*// MouseIn event - Notifies that the cursor enters the layer.*

```
private void exgauge1_MouseIn(object sender,int Layer)
{
    exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[Layer];

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedC

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGree

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueC

}
//this.exgauge1.MouseIn += new
exontrol.EXGAUGELib.exg2antt.MouseInEventHandler(this.exgauge1_MouseIn);
```

*// MouseOut event - Notifies that the cursor exits the layer.*

```
private void exgauge1_MouseOut(object sender,int Layer)
{
    exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers[Layer];

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exRedC

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exGree

    var_Layer.set_Brightness(exontrol.EXGAUGELib.ColorAdjustmentChannelEnum.exBlueC

}
//this.exgauge1.MouseOut += new
exontrol.EXGAUGELib.exg2antt.MouseOutEventHandler(this.exgauge1_MouseOut);

exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayC
```

```
exgauge1.PicturesPath = "C:\\Program Files  
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
exgauge1.Layers.Count = 1;
```

## JScript/JavaScript

```
<BODY onload="Init()">  
<SCRIPT FOR="Gauge1" EVENT="MouseIn(Layer)" LANGUAGE="JScript">  
    var var_Layer = Gauge1.Layers.Item(Layer);  
    var_Layer.Brightness(1) = 100;  
    var_Layer.Brightness(2) = 0;  
    var_Layer.Brightness(3) = 0;  
</SCRIPT>  
  
<SCRIPT FOR="Gauge1" EVENT="MouseOut(Layer)" LANGUAGE="JScript">  
    var var_Layer = Gauge1.Layers.Item(Layer);  
    var_Layer.Brightness(1) = Gauge1.DefaultLayer(128);  
    var_Layer.Brightness(2) = Gauge1.DefaultLayer(128);  
    var_Layer.Brightness(3) = Gauge1.DefaultLayer(128);  
</SCRIPT>  
  
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"  
id="Gauge1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Gauge1.DefaultLayer(128) = 51;  
    Gauge1.PicturesPath = "C:\\Program Files  
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";  
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";  
    Gauge1.Layers.Count = 1;  
}  
</SCRIPT>
```

</BODY>

## VBScript

```
<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_MouseIn(Layer)
  With Gauge1
    With .Layers.Item(Layer)
      .Brightness(1) = 100
      .Brightness(2) = 0
      .Brightness(3) = 0
    End With
  End With
End Function
</SCRIPT>
```

```
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_MouseOut(Layer)
  With Gauge1
    With .Layers.Item(Layer)
      .Brightness(1) = Gauge1.DefaultLayer(128)
      .Brightness(2) = Gauge1.DefaultLayer(128)
      .Brightness(3) = Gauge1.DefaultLayer(128)
    End With
  End With
End Function
</SCRIPT>
```

```
<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Gauge1
    .DefaultLayer(128) = 51
```

```

        .PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = "`Layer` + int(value + 1) + `.png`"
        .Layers.Count = 1
    End With
End Function
</SCRIPT>
</BODY>

```

## C# for /COM

```

// MouseIn event - Notifies that the cursor enters the layer.
private void axGauge1_MouseIn(object sender,
AxEXGAUGELib._IGaugeEvents_MouseInEvent e)
{
    EXGAUGELib.Layer var_Layer = axGauge1.Layers[e.layer];

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel,1

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChanne

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel,C

}
//this.axGauge1.MouseIn += new
AxEXGAUGELib._IGaugeEvents_MouseInEventHandler(this.axGauge1_MouseIn);

// MouseOut event - Notifies that the cursor exits the layer.
private void axGauge1_MouseOut(object sender,
AxEXGAUGELib._IGaugeEvents_MouseOutEvent e)
{
    EXGAUGELib.Layer var_Layer = axGauge1.Layers[e.layer];

    var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel,a

```

```

var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel, a
var_Layer.set_Brightness(EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel, a
}
//this.axGauge1.MouseOut += new
AxEXGAUGELib._IGaugeEvents_MouseOutEventHandler(this.axGauge1_MouseOut);

axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightn

axGauge1.PicturesPath = "C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 1;

```

## X++ (Dynamics Ax 2009)

```

// MouseIn event - Notifies that the cursor enters the layer.
void onEvent_MouseIn(int _Layer)
{
    COM com_Layer;
    anytype var_Layer;
    ;
    var_Layer = COM::createFromObject(exgauge1.Layers()).Item(_Layer); com_Layer =
var_Layer;
    com_Layer.Brightness(1/*exRedChannel*/,100);
    com_Layer.Brightness(2/*exGreenChannel*/,0);
    com_Layer.Brightness(3/*exBlueChannel*/,0);
}

// MouseOut event - Notifies that the cursor exits the layer.
void onEvent_MouseOut(int _Layer)
{

```



```

COM com_Layer;
anytype var_Layer;
;
var_Layer = COM::createFromObject(exgauge1.Layers()).Item(_Layer); com_Layer =
var_Layer;

com_Layer.Brightness(1/*exRedChannel*/,exgauge1.DefaultLayer(128/*exDefLayerBrig

com_Layer.Brightness(2/*exGreenChannel*/,exgauge1.DefaultLayer(128/*exDefLayerB

com_Layer.Brightness(3/*exBlueChannel*/,exgauge1.DefaultLayer(128/*exDefLayerBrig

}

public void init()
{
    COM com_Layer;
    anytype var_Layer;
    ;

    super();

exgauge1.DefaultLayer(128/*exDefLayerBrightness*/,COMVariant::createFromInt(51));
    exgauge1.PicturesPath("C:\\Program Files
(x86)\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(1);
}

```

## Delphi 8 (.NET only)

```

// MouseIn event - Notifies that the cursor enters the layer.
procedure TForm1.AxGauge1_MouseIn(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_MouseInEvent);

```

```

begin
  with AxGauge1 do
    begin
      with Layers.Item[TObject(e.layer)] do
        begin
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel] :=
100;
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel] :=
0;
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel] := 0;
        end;
      end
    end;
  end;

// MouseOut event - Notifies that the cursor exits the layer.
procedure TWinForm1.AxGauge1_MouseOut(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_MouseOutEvent);
begin
  with AxGauge1 do
    begin
      with Layers.Item[TObject(e.layer)] do
        begin
          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exRedChannel] :=
AxGauge1.DefaultLayer[EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness];

          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exGreenChannel] :=
AxGauge1.DefaultLayer[EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness];

          Brightness[EXGAUGELib.ColorAdjustmentChannelEnum.exBlueChannel] :=
AxGauge1.DefaultLayer[EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness];

        end;
      end
    end;
  end;

  with AxGauge1 do
    begin

```

```
set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerBrightness,TObject
```

```
    PicturesPath := 'C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';  
    PicturesName := 'Layer` + int(value + 1) + `.png`';  
    Layers.Count := 1;  
end
```

## Delphi (standard)

```
// MouseIn event - Notifies that the cursor enters the layer.
```

```
procedure TForm1.Gauge1MouseIn(ASender: TObject; Layer : Integer);  
begin  
    with Gauge1 do  
    begin  
        with Layers.Item[OleVariant(Layer)] do  
        begin  
            Brightness[EXGAUGELib_TLB.exRedChannel] := 100;  
            Brightness[EXGAUGELib_TLB.exGreenChannel] := 0;  
            Brightness[EXGAUGELib_TLB.exBlueChannel] := 0;  
        end;  
    end  
end;
```

```
// MouseOut event - Notifies that the cursor exits the layer.
```

```
procedure TForm1.Gauge1MouseOut(ASender: TObject; Layer : Integer);  
begin  
    with Gauge1 do  
    begin  
        with Layers.Item[OleVariant(Layer)] do  
        begin  
            Brightness[EXGAUGELib_TLB.exRedChannel] :=  
Gauge1.DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness];  
            Brightness[EXGAUGELib_TLB.exGreenChannel] :=  
Gauge1.DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness];  
            Brightness[EXGAUGELib_TLB.exBlueChannel] :=
```

```

Gauge1.DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness];
    end;
end
end;

with Gauge1 do
begin
    DefaultLayer[EXGAUGELib_TLB.exDefLayerBrightness] := OleVariant(51);
    PicturesPath := 'C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 1;
end

```

## VFP

*\*\*\* MouseIn event - Notifies that the cursor enters the layer. \*\*\**

```

LPARAMETERS Layer
with thisform.Gauge1
    with .Layers.Item(Layer)
        .Brightness(1) = 100
        .Brightness(2) = 0
        .Brightness(3) = 0
    endwith
endwith

```

*\*\*\* MouseOut event - Notifies that the cursor exits the layer. \*\*\**

```

LPARAMETERS Layer
with thisform.Gauge1
    with .Layers.Item(Layer)
        .Brightness(1) = thisform.Gauge1.DefaultLayer(128)
        .Brightness(2) = thisform.Gauge1.DefaultLayer(128)
        .Brightness(3) = thisform.Gauge1.DefaultLayer(128)
    endwith
endwith

```

```

with thisform.Gauge1

```

```

.Object.DefaultLayer(128) = 51
.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
.PicturesName = "\"Layer` + int(value + 1) + `.png\""
.Layers.Count = 1
endwith

```

## dBASE Plus

```

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    MouseIn = class::nativeObject_MouseIn
endwith
*/
// Notifies that the cursor enters the layer.
function nativeObject_MouseIn(Layer)
    local var_Layer
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    var_Layer = oGauge.Layers.Item(Layer)
    // var_Layer.Brightness(1) = 100
    with (oGauge)
        TemplateDef = [dim var_Layer]
        TemplateDef = var_Layer
        Template = [var_Layer.Brightness(1) = 100]
    endwith
    // var_Layer.Brightness(2) = 0
    with (oGauge)
        TemplateDef = [dim var_Layer]
        TemplateDef = var_Layer
        Template = [var_Layer.Brightness(2) = 0]
    endwith
    // var_Layer.Brightness(3) = 0
    with (oGauge)
        TemplateDef = [dim var_Layer]
        TemplateDef = var_Layer
        Template = [var_Layer.Brightness(3) = 0]
    endwith

```

return

*/\*  
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)*

*MouseOut = class::nativeObject\_MouseOut*

*endwith*

*\*/*

*// Notifies that the cursor exits the layer.*

function nativeObject\_MouseOut(Layer)

local var\_Layer

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject

var\_Layer = oGauge.Layers.Item(Layer)

*// var\_Layer.Brightness(1) = oGauge.DefaultLayer(128)*

with (oGauge)

TemplateDef = [dim var\_Layer]

TemplateDef = var\_Layer

Template = [var\_Layer.Brightness(1) = Me.DefaultLayer(128)]

endwith

*// var\_Layer.Brightness(2) = oGauge.DefaultLayer(128)*

with (oGauge)

TemplateDef = [dim var\_Layer]

TemplateDef = var\_Layer

Template = [var\_Layer.Brightness(2) = Me.DefaultLayer(128)]

endwith

*// var\_Layer.Brightness(3) = oGauge.DefaultLayer(128)*

with (oGauge)

TemplateDef = [dim var\_Layer]

TemplateDef = var\_Layer

Template = [var\_Layer.Brightness(3) = Me.DefaultLayer(128)]

endwith

return

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject

oGauge.Template = [DefaultLayer(128) = 51] *// oGauge.DefaultLayer(128) = 51*

oGauge.PicturesPath = "C:\Program Files

```
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"  
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"  
oGauge.Layers.Count = 1
```

## XBasic (Alpha Five)

*' Notifies that the cursor enters the layer.*

```
function MouseIn as v (Layer as N)  
    Dim var_Layer as P  
    oGauge = topparent:CONTROL_ACTIVEX1.activex  
    var_Layer = oGauge.Layers.Item(Layer)  
    ' var_Layer.Brightness(1) = 100  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(1) = 100"  
    ' var_Layer.Brightness(2) = 0  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(2) = 0"  
    ' var_Layer.Brightness(3) = 0  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(3) = 0"
```

end function

*' Notifies that the cursor exits the layer.*

```
function MouseOut as v (Layer as N)  
    Dim var_Layer as P  
    oGauge = topparent:CONTROL_ACTIVEX1.activex  
    var_Layer = oGauge.Layers.Item(Layer)  
    ' var_Layer.Brightness(1) = oGauge.DefaultLayer(128)  
    oGauge.TemplateDef = "dim var_Layer"  
    oGauge.TemplateDef = var_Layer  
    oGauge.Template = "var_Layer.Brightness(1) = Me.DefaultLayer(128)"  
    ' var_Layer.Brightness(2) = oGauge.DefaultLayer(128)
```

```

oGauge.TemplateDef = "dim var_Layer"
oGauge.TemplateDef = var_Layer
oGauge.Template = "var_Layer.Brightness(2) = Me.DefaultLayer(128)"
' var_Layer.Brightness(3) = oGauge.DefaultLayer(128)
oGauge.TemplateDef = "dim var_Layer"
oGauge.TemplateDef = var_Layer
oGauge.Template = "var_Layer.Brightness(3) = Me.DefaultLayer(128)"

end function

Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.Template = "DefaultLayer(128) = 51" // oGauge.DefaultLayer(128) = 51
oGauge.PicturesPath = "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = ""Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 1

```

## Visual Objects

```

METHOD OCX_Exontrol1MouseIn(Layer) CLASS MainDialog
    // MouseIn event - Notifies that the cursor enters the layer.
    local var_Layer as ILayer
    var_Layer := oDCOCX_Exontrol1:Layers:[Item,Layer]
    var_Layer:[Brightness,exRedChannel] := 100
    var_Layer:[Brightness,exGreenChannel] := 0
    var_Layer:[Brightness,exBlueChannel] := 0
RETURN NIL

METHOD OCX_Exontrol1MouseOut(Layer) CLASS MainDialog
    // MouseOut event - Notifies that the cursor exits the layer.
    local var_Layer as ILayer
    var_Layer := oDCOCX_Exontrol1:Layers:[Item,Layer]
    var_Layer:[Brightness,exRedChannel] := oDCOCX_Exontrol1:
[DefaultLayer,exDefLayerBrightness]

```



```

    var_Layer:[Brightness,exGreenChannel] := oDCOCX_Exontrol1:
[DefaultLayer,exDefLayerBrightness]
    var_Layer:[Brightness,exBlueChannel] := oDCOCX_Exontrol1:
[DefaultLayer,exDefLayerBrightness]
RETURN NIL

oDCOCX_Exontrol1:[DefaultLayer,exDefLayerBrightness] := 51
oDCOCX_Exontrol1:PicturesPath := "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers:Count := 1

```

## PowerBuilder

```

/*begin event MouseIn(long Layer) - Notifies that the cursor enters the layer.*/
/*
    OleObject var_Layer
    oGauge = ole_1.Object
    var_Layer = oGauge.Layers.Item(Layer)
        var_Layer.Brightness(1,100)
        var_Layer.Brightness(2,0)
        var_Layer.Brightness(3,0)
*/
/*end event MouseIn*/

/*begin event MouseOut(long Layer) - Notifies that the cursor exits the layer.*/
/*
    OleObject var_Layer
    oGauge = ole_1.Object
    var_Layer = oGauge.Layers.Item(Layer)
        var_Layer.Brightness(1,oGauge.DefaultLayer(128))
        var_Layer.Brightness(2,oGauge.DefaultLayer(128))
        var_Layer.Brightness(3,oGauge.DefaultLayer(128))
*/
/*end event MouseOut*/

```

OleObject oGauge

oGauge = ole\_1.Object

oGauge.DefaultLayer(128,51)

oGauge.PicturesPath = "C:\Program Files  
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"

oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"

oGauge.Layers.Count = 1

## Visual DataFlex

*// Notifies that the cursor enters the layer.*

Procedure OnComMouseIn Integer IILayer

Forward Send OnComMouseIn IILayer

Variant voLayers

Get ComLayers to voLayers

Handle hoLayers

Get Create (RefClass(cComLayers)) to hoLayers

Set pvComObject of hoLayers to voLayers

Variant voLayer

Get ComItem of hoLayers IILayer to voLayer

Handle hoLayer

Get Create (RefClass(cComLayer)) to hoLayer

Set pvComObject of hoLayer to voLayer

Set **ComBrightness** of hoLayer OLEexRedChannel to 100

Set **ComBrightness** of hoLayer OLEexGreenChannel to 0

Set **ComBrightness** of hoLayer OLEexBlueChannel to 0

Send Destroy to hoLayer

Send Destroy to hoLayers

End\_Procedure

*// Notifies that the cursor exits the layer.*

Procedure OnComMouseOut Integer IILayer

Forward Send OnComMouseOut IILayer

Variant voLayers1

```

Get ComLayers to voLayers1
Handle hoLayers1
Get Create (RefClass(cComLayers)) to hoLayers1
Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer1
    Get ComItem of hoLayers1 llLayer to voLayer1
    Handle hoLayer1
    Get Create (RefClass(cComLayer)) to hoLayer1
    Set pvComObject of hoLayer1 to voLayer1
        Variant v
            Get ComDefaultLayer OLEexDefLayerBrightness to v
            Set ComBrightness of hoLayer1 OLEexRedChannel to v
            Variant v1
                Get ComDefaultLayer OLEexDefLayerBrightness to v1
                Set ComBrightness of hoLayer1 OLEexGreenChannel to v1
                Variant v2
                    Get ComDefaultLayer OLEexDefLayerBrightness to v2
                    Set ComBrightness of hoLayer1 OLEexBlueChannel to v2
            Send Destroy to hoLayer1
        Send Destroy to hoLayers1
End_Procedure

```

#### Procedure OnCreate

```

Forward Send OnCreate
Set ComDefaultLayer OLEexDefLayerBrightness to 51
Set ComPicturesPath to "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers2
Get ComLayers to voLayers2
Handle hoLayers2
Get Create (RefClass(cComLayers)) to hoLayers2
Set pvComObject of hoLayers2 to voLayers2
    Set ComCount of hoLayers2 to 1
    Send Destroy to hoLayers2
End_Procedure

```

```
PROCEDURE OnMouseIn(oGauge,Layer)
```

```
    LOCAL oLayer
```

```
    oLayer := oGauge:Layers:Item(Layer)
```

```
        oLayer:SetProperty("Brightness",1/*exRedChannel*/,100)
```

```
        oLayer:SetProperty("Brightness",2/*exGreenChannel*/,0)
```

```
        oLayer:SetProperty("Brightness",3/*exBlueChannel*/,0)
```

```
RETURN
```

```
PROCEDURE OnMouseOut(oGauge,Layer)
```

```
    LOCAL oLayer
```

```
    oLayer := oGauge:Layers:Item(Layer)
```

```
oLayer:SetProperty("Brightness",1/*exRedChannel*/,oGauge:DefaultLayer(128/*exDefL
```

```
oLayer:SetProperty("Brightness",2/*exGreenChannel*/,oGauge:DefaultLayer(128/*exDe
```

```
oLayer:SetProperty("Brightness",3/*exBlueChannel*/,oGauge:DefaultLayer(128/*exDefl
```

```
RETURN
```

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oGauge
```

```
oForm := XbpDialog():new( AppDesktop() )
```

```
oForm:drawingArea:clipChildren := .T.
```

```
oForm:create( „{100,100}, {640,480},„ .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```

oGauge := XbpActiveXControl():new( oForm:drawingArea )
oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
oGauge:create(,, {10,60},{610,370} )

    oGauge:MouseIn := {|Layer| OnMouseIn(oGauge,Layer)} /*Notifies that the
cursor enters the layer.*/
    oGauge:MouseOut := {|Layer| OnMouseOut(oGauge,Layer)} /*Notifies that the
cursor exits the layer.*/

    oGauge:SetProperty("DefaultLayer",128/*exDefLayerBrightness*/,51)
    oGauge:PicturesPath := "C:\Program Files
(x86)\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
    oGauge:Layers():Count := 1

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

## event MouseUp (Button as Integer, Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS)

Occurs when the user releases a mouse button.

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

Use a [MouseDown](#) or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The Click event is not fired if you click, drag and release the mouse over the control. The [OnDrag](#) property indicates the action to be performed when the user clicks and drags the layer.

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,
```

```
AxEXGAUGELib._IGaugeEvents_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: AxEXGAUGELib._IGaugeEvents_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 AxEXGAUGELib._IGaugeEvents_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(oGauge,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.Gauge.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Gauge.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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



# event MouseWheel (Delta as Long)

Occurs when the mouse wheel moves while the control has focus

Type	Description
Delta as Long	A long expression that specifies the direction and the quantity that the mouse wheel has been rolled. For instance, 1 indicates that the user rolls the mouse wheel up, -1 indicates that the user rolls the mouse wheel down. Any other value may indicate that the mouse wheel has been rolled quicker.

The MouseWheel occurs when the mouse wheel is rolled. You can use the MouseWheel event to perform different actions on any layer when the user rolls the mouse wheel. For instance, you can update the layer's value when the user rolls the mouse wheel. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The [FormatABC](#) method formats the A,B,C values based on the giving expression and returns the result.

You can use any of the following properties to update the layer:

- [Value](#), specifies the layer's value.
- [OffsetX](#), specifies a value that indicates x-offset of the layer.
- [OffsetY](#), indicates a value that indicates y-offset of the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [Clip](#), to clip any layer

The [Change](#) event occurs when the layer's value is changed.

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

```
C# private void MouseWheel(object sender,int  Delta)
{
}
```

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

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

```
C# private void MouseWheel(object sender,
AxEXGAUGELib._IGaugeEvents_MouseWheelEvent e)
{
```

```
}
```

C++

```
void OnMouseWheel(long Delta)
{
}
```

C++  
Builder

```
void __fastcall MouseWheel(TObject *Sender,long Delta)
{
}
```

Delphi

```
procedure MouseWheel(ASender: TObject; Delta : Integer);
begin
end;
```

Delphi 8  
(.NET  
only)

```
procedure MouseWheel(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_MouseWheelEvent);
begin
end;
```

Power...

```
begin event MouseWheel(long Delta)

end event MouseWheel
```

VB.NET

```
Private Sub MouseWheel(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_MouseWheelEvent) Handles MouseWheel
End Sub
```

VB6

```
Private Sub MouseWheel(ByVal Delta As Long)
End Sub
```

VBA

```
Private Sub MouseWheel(ByVal Delta As Long)
End Sub
```

VFP

```
LPARAMETERS Delta
```

Xbas...

```
PROCEDURE OnMouseWheel(oGauge,Delta)
```

## RETURN

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

Java... <SCRIPT EVENT="MouseWheel(Delta)" LANGUAGE="JScript">  
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">  
Function MouseWheel(Delta)  
End Function  
</SCRIPT>

Visual  
Data... Procedure OnComMouseWheel Integer IIDelta  
Forward Send OnComMouseWheel IIDelta  
End\_Procedure

Visual  
Objects METHOD OCX\_MouseWheel(Delta) CLASS MainDialog  
RETURN NIL

X++ void onEvent\_MouseWheel(int \_Delta)  
{  
}

XBasic function MouseWheel as v (Delta as N)  
end function

dBASE function nativeObject\_MouseWheel(Delta)  
return

The following sample rotates the first visible layer by 15% degrees ( up / down ), when the user rolls the mouse wheel:

### VBA (MS Access, Excell...)

*' MouseWheel event - Occurs when the mouse wheel moves while the control has focus*  
Private Sub Gauge1\_MouseWheel(ByVal Delta As Long)  
With Gauge1

```

    With .Layers.Item("rotateOnWheel")
        .RotateAngle = Gauge1.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
        Debug.Print( .RotateAngle )
    End With
End With
End Sub

With Gauge1
    .DefaultLayer(185) = 1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
    .Layers.Item(0).Key = "rotateOnWheel"
End With

```

## VB6

*' MouseWheel event - Occurs when the mouse wheel moves while the control has focus*

```

Private Sub Gauge1_MouseWheel(ByVal Delta As Long)
    With Gauge1
        With .Layers.Item("rotateOnWheel")
            .RotateAngle = Gauge1.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
            Debug.Print( .RotateAngle )
        End With
    End With
End Sub

```

```

With Gauge1
    .DefaultLayer(exDefLayerRotateType) = 1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11

```

```
.Layers.Item(0).Key = "rotateOnWheel"  
End With
```

## VB.NET

*' MouseWheel event - Occurs when the mouse wheel moves while the control has focus*

```
Private Sub Exgauge1_MouseWheel(ByVal sender As System.Object, ByVal Delta As Integer) Handles Exgauge1.MouseWheel
```

```
    With Exgauge1
```

```
        With .Layers.Item("rotateOnWheel")
```

```
            .RotateAngle = Exgauge1.FormatABC("A + (15 *  
B)", .Layers.Item("rotateOnWheel").RotateAngle, Delta)
```

```
            Debug.Print( .RotateAngle )
```

```
        End With
```

```
    End With
```

```
End Sub
```

```
With Exgauge1
```

```
.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateTy
```

```
.PicturesPath = "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
.PicturesName = ""Layer` + int(value + 1) + `.png"
```

```
.Layers.Count = 11
```

```
.Layers.Item(0).Key = "rotateOnWheel"
```

```
End With
```

## VB.NET for /COM

*' MouseWheel event - Occurs when the mouse wheel moves while the control has focus*

```
Private Sub AxGauge1_MouseWheel(ByVal sender As System.Object, ByVal e As AxEXGAUGELib._IGaugeEvents_MouseWheelEvent) Handles AxGauge1.MouseWheel
```

```
    With AxGauge1
```

```
        With .Layers.Item("rotateOnWheel")
```

```
            .RotateAngle = AxGauge1.FormatABC("A + (15 *
```

```

B)".Layers.Item("rotateOnWheel").RotateAngle,e.delta)
    Debug.Print( .RotateAngle )
End With
End With
End Sub

With AxGauge1
    .set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,1)
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
    .Layers.Item(0).Key = "rotateOnWheel"
End With

```

## C++

```

// MouseWheel event - Occurs when the mouse wheel moves while the control has
focus
void OnMouseWheelGauge1(long Delta)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control
        Library'
        #import <ExGauge.dll>
        using namespace EXGAUGELib;
    */
    EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
    EXGAUGELib::ILayerPtr var_Layer = spGauge1->GetLayers()-
> GetItem("rotateOnWheel");
    var_Layer->PutRotateAngle(spGauge1->FormatABC(L"A + (15 * B)",-
> GetLayers()-> GetItem("rotateOnWheel")->GetRotateAngle(),Delta,vtMissing));
    OutputDebugStringW( _bstr_t(var_Layer->GetRotateAngle()) );
}

```

```

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC_GAUGE1)-
> GetControlUnknown();
spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerRotateType,long(1));
spGauge1->PutPicturesPath(L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(11);
spGauge1->GetLayers()->GetItem(long(0))->PutKey("rotateOnWheel");

```

## C++ Builder

*// MouseWheel event - Occurs when the mouse wheel moves while the control has focus*

```

void __fastcall TForm1::Gauge1MouseWheel(TObject *Sender,long Delta)
{
    Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers-
> get_Item(TVariant("rotateOnWheel"));
    var_Layer->RotateAngle = ->FormatABC(L"A + (15 * B)",TVariant(Gauge1-
> Layers-> get_Item(TVariant("rotateOnWheel"))-
> RotateAngle),TVariant(Delta),TNoParam());
    OutputDebugString( PChar(var_Layer->RotateAngle) );
}

```

```

Gauge1-
> DefaultLayer[Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerRotateType] =
TVariant(1);
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png";
Gauge1->Layers->Count = 11;
Gauge1->Layers->get_Item(TVariant(0))->set_Key(TVariant("rotateOnWheel"));

```

## C#

*// MouseWheel event - Occurs when the mouse wheel moves while the control has focus*

```

private void exgauge1_MouseWheel(object sender,int Delta)
{
    exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers["rotateOnWheel"];
    var_Layer.RotateAngle = exgauge1.FormatABC("A + (15 *
B)",.Layers["rotateOnWheel"].RotateAngle,Delta,null);
    System.Diagnostics.Debug.Print( var_Layer.RotateAngle.ToString() );
}
//this.exgauge1.MouseWheel += new
exontrol.EXGAUGELib.exg2antt.MouseWheelEventHandler(this.exgauge1_MouseWheel);

exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLay

exgauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
exgauge1.Layers.Count = 11;
exgauge1.Layers[0].Key = "rotateOnWheel";

```

## JScript/JavaScript

```

<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="MouseWheel(Delta)" LANGUAGE="JScript">
    var var_Layer = Gauge1.Layers.Item("rotateOnWheel");
    var_Layer.RotateAngle = Gauge1.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta,null);
    alert( var_Layer.RotateAngle );
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.DefaultLayer(185) = 1;

```



```

Gauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
Gauge1.Layers.Count = 11;
Gauge1.Layers.Item(0).Key = "rotateOnWheel";
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_MouseWheel(Delta)
    With Gauge1
        With .Layers.Item("rotateOnWheel")
            .RotateAngle = Gauge1.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
            alert( .RotateAngle )
        End With
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .DefaultLayer(185) = 1
        .PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob"
        .PicturesName = "`Layer` + int(value + 1) + `.png`"
        .Layers.Count = 11
        .Layers.Item(0).Key = "rotateOnWheel"
    End With
End Function

```

```
End With
End Function
</SCRIPT>
</BODY>
```

## C# for /COM

```
// MouseWheel event - Occurs when the mouse wheel moves while the control has focus
private void axGauge1_MouseWheel(object sender,
AxEXGAUGELib._IGaugeEvents_MouseWheelEvent e)
{
    EXGAUGELib.Layer var_Layer = axGauge1.Layers["rotateOnWheel"];
    var_Layer.RotateAngle = axGauge1.FormatABC("A + (15 *
B)",.Layers["rotateOnWheel"].RotateAngle,e.delta,null);
    System.Diagnostics.Debug.Print( var_Layer.RotateAngle.ToString() );
}
//this.axGauge1.MouseWheel += new
AxEXGAUGELib._IGaugeEvents_MouseWheelEventHandler(this.axGauge1_MouseWheel);

axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotate

axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 11;
axGauge1.Layers[0].Key = "rotateOnWheel";
```

## X++ (Dynamics Ax 2009)

```
// MouseWheel event - Occurs when the mouse wheel moves while the control has focus
void onEvent_MouseWheel(int _Delta)
{
    COM com_Layer;
```

```

anytype var_Layer;
;
var_Layer = COM::createFromObject(exgauge1.Layers()).Item("rotateOnWheel");
com_Layer = var_Layer;
    com_Layer.RotateAngle(exgauge1.FormatABC("A + (15 *
B)",Layers().Item("rotateOnWheel").RotateAngle(),_Delta));
    print( com_Layer.RotateAngle() );
}

public void init()
{
    COM com_Layer;
    anytype var_Layer;
    ;

    super();

exgauge1.DefaultLayer(185/*exDefLayerRotateType*/,COMVariant::createFromInt(1));
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(11);
    var_Layer =
COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(0));
com_Layer = var_Layer;
    com_Layer.Key("rotateOnWheel");
}

```

## Delphi 8 (.NET only)

```

// MouseWheel event - Occurs when the mouse wheel moves while the control has focus
procedure TForm1.AxGauge1_MouseWheel(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_MouseWheelEvent);
begin
    with AxGauge1 do

```

```

begin
  with Layers.Item['rotateOnWheel'] do
    begin
      RotateAngle := .FormatABC('A + (15 *
B)',TObject(AxGauge1.Layers.Item['rotateOnWheel'].RotateAngle),TObject(e.delta),Nil);
      OutputDebugString( RotateAngle );
    end;
  end
end;

with AxGauge1 do
begin

set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,TObje

  PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
  PicturesName := 'Layer` + int(value + 1) + `.png`;
  Layers.Count := 11;
  Layers.Item[TObject(0)].Key := 'rotateOnWheel';
end

```

## Delphi (standard)

```

// MouseWheel event - Occurs when the mouse wheel moves while the control has focus
procedure TForm1.Gauge1MouseWheel(ASender: TObject; Delta : Integer);
begin
  with Gauge1 do
    begin
      with Layers.Item['rotateOnWheel'] do
        begin
          RotateAngle := .FormatABC('A + (15 *
B)',OleVariant(Gauge1.Layers.Item['rotateOnWheel'].RotateAngle),OleVariant(Delta),Nul

          OutputDebugString( RotateAngle );
        end;
      end;
    end;
  end;
end;

```

```

    end
end;

with Gauge1 do
begin
    DefaultLayer[EXGAUGELib_TLB.exDefLayerRotateType] := OleVariant(1);
    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := '`Layer` + int(value + 1) + `.png`';
    Layers.Count := 11;
    Layers.Item[OleVariant(0)].Key := 'rotateOnWheel';
end

```

## VFP

*\*\*\* MouseWheel event - Occurs when the mouse wheel moves while the control has focus \*\*\**

```

LPARAMETERS Delta
with thisform.Gauge1
    with .Layers.Item("rotateOnWheel")
        .RotateAngle = thisform.Gauge1.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
        DEBUGOUT( .RotateAngle )
    endwith
endwith

with thisform.Gauge1
    .Object.DefaultLayer(185) = 1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
    .Layers.Item(0).Key = "rotateOnWheel"
endwith

```

## dBASE Plus

/\*

```

with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    MouseWheel = class::nativeObject_MouseWheel
endwith
*/
// Occurs when the mouse wheel moves while the control has focus
function nativeObject_MouseWheel(Delta)
    local var_Layer
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    var_Layer = oGauge.Layers.Item("rotateOnWheel")
    var_Layer.RotateAngle = oGauge.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
    ? Str(var_Layer.RotateAngle)
return

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.Template = [DefaultLayer(185) = 1] // oGauge.DefaultLayer(185) = 1
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.Layers.Item(0).Key = "rotateOnWheel"

```

## XBasic (Alpha Five)

```

' Occurs when the mouse wheel moves while the control has focus
function MouseWheel as v (Delta as N)
    Dim var_Layer as P
    oGauge = topparent:CONTROL_ACTIVEX1.activex
    var_Layer = oGauge.Layers.Item("rotateOnWheel")
    var_Layer.RotateAngle = oGauge.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
    ? var_Layer.RotateAngle
end function

```

Dim oGauge as P

```
oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.Template = "DefaultLayer(185) = 1" // oGauge.DefaultLayer(185) = 1
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.Layers.Item(0).Key = "rotateOnWheel"
```

## Visual Objects

```
METHOD OCX_Exontrol1MouseWheel(Delta) CLASS MainDialog
    // MouseWheel event - Occurs when the mouse wheel moves while the control has
    // focus
    local var_Layer as ILayer
    var_Layer := oDCOCX_Exontrol1:Layers:[Item,"rotateOnWheel"]
    var_Layer:RotateAngle := oDCOCX_Exontrol1:FormatABC("A + (15 * B)",:Layers:
    [Item,"rotateOnWheel"]):RotateAngle,Delta,nil)
    OutputDebugString(String2Psz( AsString(var_Layer:RotateAngle) ))
RETURN NIL

oDCOCX_Exontrol1:[DefaultLayer,exDefLayerRotateType] := 1
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers.Count := 11
oDCOCX_Exontrol1:Layers:[Item,0]:Key := "rotateOnWheel"
```

## PowerBuilder

```
/*begin event MouseWheel(long Delta) - Occurs when the mouse wheel moves while
the control has focus*/
/*
    OleObject var_Layer
```

```

oGauge = ole_1.Object
var_Layer = oGauge.Layers.Item("rotateOnWheel")
    var_Layer.RotateAngle = oGauge.FormatABC("A + (15 *
B)",.Layers.Item("rotateOnWheel").RotateAngle,Delta)
    MessageBox("Information",string( String(var_Layer.RotateAngle) ))
*/
/*end event MouseWheel*/

```

OleObject oGauge

```

oGauge = ole_1.Object
oGauge.DefaultLayer(185,1)
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.Layers.Item(0).Key = "rotateOnWheel"

```

## Visual DataFlex

```

// Occurs when the mouse wheel moves while the control has focus
Procedure OnComMouseWheel Integer IIDelta
    Forward Send OnComMouseWheel IIDelta
    Variant voLayers
    Get ComLayers to voLayers
    Handle hoLayers
    Get Create (RefClass(cComLayers)) to hoLayers
    Set pvComObject of hoLayers to voLayers
    Variant voLayer
    Get ComItem of hoLayers "rotateOnWheel" to voLayer
    Handle hoLayer
    Get Create (RefClass(cComLayer)) to hoLayer
    Set pvComObject of hoLayer to voLayer
    Variant v
        Variant vA
        Variant voLayers1

```



```

    Get ComLayers to voLayers1
    Handle hoLayers1
    Get Create (RefClass(cComLayers)) to hoLayers1
    Set pvComObject of hoLayers1 to voLayers1
    Variant voLayer1
    Get ComItem of hoLayers1 "rotateOnWheel" to voLayer1
    Handle hoLayer1
    Get Create (RefClass(cComLayer)) to hoLayer1
    Set pvComObject of hoLayer1 to voLayer1
    Get ComRotateAngle of hoLayer1 to vA
    Send Destroy to hoLayer1
    Send Destroy to hoLayers1
    Get ComFormatABC "A + (15 * B)" vA IIDelta Nothing to v
    Set ComRotateAngle of hoLayer to v
    ShowIn (ComRotateAngle(hoLayer))
    Send Destroy to hoLayer
    Send Destroy to hoLayers
End_Procedure

```

#### Procedure OnCreate

```

    Forward Send OnCreate
    Set ComDefaultLayer OLEexDefLayerRotateType to 1
    Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    Set ComPicturesName to ""Layer` + int(value + 1) + `.png`"
    Variant voLayers2
    Get ComLayers to voLayers2
    Handle hoLayers2
    Get Create (RefClass(cComLayers)) to hoLayers2
    Set pvComObject of hoLayers2 to voLayers2
    Set ComCount of hoLayers2 to 11
    Send Destroy to hoLayers2
    Variant voLayers3
    Get ComLayers to voLayers3
    Handle hoLayers3
    Get Create (RefClass(cComLayers)) to hoLayers3
    Set pvComObject of hoLayers3 to voLayers3

```

```

Variant voLayer2
Get ComItem of hoLayers3 0 to voLayer2
Handle hoLayer2
Get Create (RefClass(cComLayer)) to hoLayer2
Set pvComObject of hoLayer2 to voLayer2
    Set ComKey of hoLayer2 to "rotateOnWheel"
Send Destroy to hoLayer2
Send Destroy to hoLayers3
End_Procedure

```

## XBase++

```

PROCEDURE OnMouseWheel(oGauge,Delta)
    LOCAL oLayer
    oLayer := oGauge:Layers:Item("rotateOnWheel")
    oLayer:RotateAngle := oGauge:FormatABC("A + (15 *
B)",:Layers:Item("rotateOnWheel"):RotateAngle(),Delta)
    DevOut( Transform(oLayer:RotateAngle(),"") )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

```

```
oGauge:MouseWheel := {|Delta| OnMouseWheel(oGauge,Delta)} /*Occurs when  
the mouse wheel moves while the control has focus*/
```

```
oGauge:SetProperty("DefaultLayer",185/*exDefLayerRotateType*/,1)
```

```
oGauge:PicturesPath := "C:\Program  
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
```

```
oGauge:Layers():Count := 11
```

```
oGauge:Layers:Item(0):Key := "rotateOnWheel"
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

# event RClick ()

Occurs once the user right clicks the control.

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

Use the RClick event to add your context menu. The RClick event notifies your application when the user right clicks the control. Use the [Click](#) event to notify your application that the user clicks the control ( using the left mouse button ). Use the [MouseDown](#) or [MouseUp](#) event if you require the cursor position during the RClick event. The [MouseIn](#) / [MouseOut](#) event notifies your application when the cursor is entering / leaving the layer. You can use the [LayerFromPoint\(-1,-1\)](#) property to get the layer from the cursor. The Click event is not fired if you click, drag and release the mouse over the control. The [OnDrag](#) property indicates the action to be performed when the user clicks and drags the layer.

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

C#	<pre>private void RClick(object sender) { }</pre>
VB	<pre>Private Sub RClick(ByVal sender As System.Object) Handles RClick End Sub</pre>

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

C#	<pre>private void RClick(object sender, EventArgs e) { }</pre>
C++	<pre>void OnRClick() { }</pre>
C++ Builder	<pre>void __fastcall RClick(TObject *Sender) { }</pre>
Delphi	<pre>procedure RClick(ASender: TObject; ); begin end;</pre>

Delphi 8  
(.NET only)

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

Powe...

```
begin event RClick()  
  
end event RClick
```

VB.NET

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

VB6

```
Private Sub RClick()  
End Sub
```

VBA

```
Private Sub RClick()  
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnRClick(oGauge)  
  
RETURN
```

Syntax for RClick event, **ICOM** version (others), on:

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComRClick  
    Forward Send OnComRClick  
End_Procedure
```

Visual  
Objects

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

X++

```
void onEvent_RClick()  
{  
}
```

XBasic

```
function RClick as v ()  
end function
```

dBASE

```
function nativeObject_RClick()  
return
```

# event Timer (TickCount as Long)

Occurs when the interval elapses.

Type	Description
TickCount as Long	A Long expression that specifies the number of milliseconds that have elapsed since the system was started, up to 49.7 days.

The Timer event occurs when the timer interval elapses. The [TimerInterval](#) property returns or sets the number of milliseconds between calls of control's Timer event. You can use the Timer event to perform different actions on any layer when a specified time elapsed. For instance, you can rotate the layer every second, or any dial of a clock, and so on.

The [FormatABC](#) method formats the A,B,C values based on the giving expression and returns the result.

You can use any of the following properties to update the layer:

- [Value](#), specifies the layer's value.
- [OffsetX](#), specifies a value that indicates x-offset of the layer.
- [OffsetY](#), indicates a value that indicates y-offset of the layer.
- [RotateAngle](#), specifies the angle to rotate the layer.
- [Clip](#), to clip any layer

The [Change](#) event occurs when the layer's value is changed.

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

```
C# private void Timer(object sender,int  TickCount)
{
}
```

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

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

```
C# private void Timer(object sender, AxEXGAUGELib._IGaugeEvents_TimerEvent e)
{
}
```

**C++**

```
void OnTimer(long TickCount)
{
}
```

**C++  
Builder**

```
void __fastcall Timer(TObject *Sender,long TickCount)
{
}
```

**Delphi**

```
procedure Timer(ASender: TObject; TickCount : Integer);
begin
end;
```

**Delphi 8  
(.NET  
only)**

```
procedure Timer(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_TimerEvent);
begin
end;
```

**Powe...**

```
begin event Timer(long TickCount)

end event Timer
```

**VB.NET**

```
Private Sub Timer(ByVal sender As System.Object, ByVal e As
AxEXGAUGELib._IGaugeEvents_TimerEvent) Handles Timer
End Sub
```

**VB6**

```
Private Sub Timer(ByVal TickCount As Long)
End Sub
```

**VBA**

```
Private Sub Timer(ByVal TickCount As Long)
End Sub
```

**VFP**

```
LPARAMETERS TickCount
```

**Xbas...**

```
PROCEDURE OnTimer(oGauge,TickCount)

RETURN
```



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

```
Java... <SCRIPT EVENT="Timer(TickCount)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function Timer(TickCount)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComTimer Integer lTickCount  
Forward Send OnComTimer lTickCount  
End_Procedure
```

```
Visual  
Objects METHOD OCX_Timer(TickCount) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_Timer(int _TickCount)  
{  
}  
}
```

```
XBasic function Timer as v (TickCount as N)  
end function
```

```
dBASE function nativeObject_Timer(TickCount)  
return
```

The following samples show how you can rotate the first visible layer every second:

### VBA (MS Access, Excell...)

```
' Timer event - Occurs when the interval elapses.  
Private Sub Gauge1_Timer(ByVal TickCount As Long)  
With Gauge1  
With .Layers.Item("rotateOnTimer")  
.RotateAngle = Gauge1.FormatABC("A +  
5",.Layers.Item("rotateOnTimer").RotateAngle)
```

```

        Debug.Print( TickCount )
    End With
End With
End Sub

With Gauge1
    .DefaultLayer(185) = 1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
    .Layers.Item(0).Key = "rotateOnTimer"
    .TimerInterval = 1000
End With

```

## VB6

```

' Timer event - Occurs when the interval elapses.
Private Sub Gauge1_Timer(ByVal TickCount As Long)
    With Gauge1
        With .Layers.Item("rotateOnTimer")
            .RotateAngle = Gauge1.FormatABC("A +
5",.Layers.Item("rotateOnTimer").RotateAngle)
            Debug.Print( TickCount )
        End With
    End With
End Sub

With Gauge1
    .DefaultLayer(exDefLayerRotateType) = 1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
    .Layers.Count = 11
    .Layers.Item(0).Key = "rotateOnTimer"
    .TimerInterval = 1000
End With

```

## VB.NET

*' Timer event - Occurs when the interval elapses.*

```
Private Sub Exgauge1_Timer(ByVal sender As System.Object, ByVal TickCount As Integer) Handles Exgauge1.Timer
```

```
    With Exgauge1
```

```
        With .Layers.Item("rotateOnTimer")
```

```
            .RotateAngle = Exgauge1.FormatABC("A +
```

```
5",.Layers.Item("rotateOnTimer").RotateAngle)
```

```
            Debug.Print( TickCount )
```

```
        End With
```

```
    End With
```

```
End Sub
```

```
With Exgauge1
```

```
.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateTy
```

```
    .PicturesPath = "C:\Program
```

```
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
    .PicturesName = "`Layer` + int(value + 1) + `.png`"
```

```
    .Layers.Count = 11
```

```
    .Layers.Item(0).Key = "rotateOnTimer"
```

```
    .TimerInterval = 1000
```

```
End With
```

## VB.NET for /COM

*' Timer event - Occurs when the interval elapses.*

```
Private Sub AxGauge1_Timer(ByVal sender As System.Object, ByVal e As AxEXGAUGELib._IGaugeEvents_TimerEvent) Handles AxGauge1.Timer
```

```
    With AxGauge1
```

```
        With .Layers.Item("rotateOnTimer")
```

```
            .RotateAngle = AxGauge1.FormatABC("A +
```

```
5",.Layers.Item("rotateOnTimer").RotateAngle)
```

```
            Debug.Print( e.tickCount )
```

```
        End With
```

```
    End With
```

End Sub

With AxGauge1

.set\_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,1)

.PicturesPath = "C:\Program

Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"

.PicturesName = ""Layer` + int(value + 1) + `.png`"

.Layers.Count = 11

.Layers.Item(0).Key = "rotateOnTimer"

.TimerInterval = 1000

End With

C++

*// Timer event - Occurs when the interval elapses.*

void OnTimerGauge1(long TickCount)

{

*/\**

*Copy and paste the following directives to your header file as*

*it defines the namespace 'EXGAUGELib' for the library: 'ExGauge 1.0 Control*

*Library'*

*#import <ExGauge.dll>*

*using namespace EXGAUGELib;*

*\*/*

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC\_GAUGE1)-

> GetControlUnknown();

EXGAUGELib::ILayerPtr var\_Layer = spGauge1->GetLayers()-

> GetItem("rotateOnTimer");

var\_Layer->PutRotateAngle(spGauge1->FormatABC(L"A + 5",->GetLayers()-

> GetItem("rotateOnTimer")->GetRotateAngle(),vtMissing,vtMissing));

OutputDebugStringW( L"TickCount" );

}

EXGAUGELib::IGaugePtr spGauge1 = GetDlgItem(IDC\_GAUGE1)-

> GetControlUnknown();

spGauge1->PutDefaultLayer(EXGAUGELib::exDefLayerRotateType,long(1));

spGauge1->PutPicturesPath(L"C:\\Program

```
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
spGauge1->PutPicturesName(L"Layer` + int(value + 1) + `.png");
spGauge1->GetLayers()->PutCount(11);
spGauge1->GetLayers()->GetItem(long(0))->PutKey("rotateOnTimer");
spGauge1->PutTimerInterval(1000);
```

## C++ Builder

*// Timer event - Occurs when the interval elapses.*

```
void __fastcall TForm1::Gauge1Timer(TObject *Sender,long TickCount)
{
    Exgaugelib_tlb::ILayerPtr var_Layer = Gauge1->Layers-
>get_Item(TVariant("rotateOnTimer"));
    var_Layer->RotateAngle = ->FormatABC(L"A + 5",TVariant(Gauge1->Layers-
>get_Item(TVariant("rotateOnTimer"))->RotateAngle),TNoParam(),TNoParam());
    OutputDebugString( L"TickCount" );
}

Gauge1-
>DefaultLayer[Exgaugelib_tlb::DefaultLayerPropertyEnum::exDefLayerRotateType] =
TVariant(1);
Gauge1->PicturesPath = L"C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
Gauge1->PicturesName = L"Layer` + int(value + 1) + `.png";
Gauge1->Layers->Count = 11;
Gauge1->Layers->get_Item(TVariant(0))->set_Key(TVariant("rotateOnTimer"));
Gauge1->TimerInterval = 1000;
```

## C#

*// Timer event - Occurs when the interval elapses.*

```
private void exgauge1_Timer(object sender,int TickCount)
{
    exontrol.EXGAUGELib.Layer var_Layer = exgauge1.Layers["rotateOnTimer"];
    var_Layer.RotateAngle = exgauge1.FormatABC("A +
5",.Layers["rotateOnTimer"].RotateAngle,null,null);
```

```

        System.Diagnostics.Debug.Print( TickCount.ToString() );
    }
    //this.exgauge1.Timer += new
    exontrol.EXGAUGELib.exg2antt.TimerEventHandler(this.exgauge1_Timer);

    exgauge1.set_DefaultLayer(exontrol.EXGAUGELib.DefaultLayerPropertyEnum.exDefLayer

    exgauge1.PicturesPath = "C:\\Program
    Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    exgauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    exgauge1.Layers.Count = 11;
    exgauge1.Layers[0].Key = "rotateOnTimer";
    exgauge1.TimerInterval = 1000;

```

## JScript/JavaScript

```

<BODY onload="Init()">
<SCRIPT FOR="Gauge1" EVENT="Timer(TickCount)" LANGUAGE="JScript">
    var var_Layer = Gauge1.Layers.Item("rotateOnTimer");
    var_Layer.RotateAngle = Gauge1.FormatABC("A +
    5",Layers.Item("rotateOnTimer").RotateAngle,null,null);
    alert( TickCount );
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Gauge1.DefaultLayer(185) = 1;
    Gauge1.PicturesPath = "C:\\Program
    Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
    Gauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
    Gauge1.Layers.Count = 11;
    Gauge1.Layers.Item(0).Key = "rotateOnTimer";

```

```

    Gauge1.TimerInterval = 1000;
}
</SCRIPT>
</BODY>

```

## VBScript

```

<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Gauge1_Timer(TickCount)
    With Gauge1
        With .Layers.Item("rotateOnTimer")
            .RotateAngle = Gauge1.FormatABC("A +
5",.Layers.Item("rotateOnTimer").RotateAngle)
            alert( TickCount )
        End With
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:91628F12-393C-44EF-A558-83ED1790AAD3"
id="Gauge1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Gauge1
        .DefaultLayer(185) = 1
        .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
        .PicturesName = ""Layer` + int(value + 1) + `.png`"
        .Layers.Count = 11
        .Layers.Item(0).Key = "rotateOnTimer"
        .TimerInterval = 1000
    End With
End Function
</SCRIPT>

```

</BODY>

## C# for /COM

```
// Timer event - Occurs when the interval elapses.
private void axGauge1_Timer(object sender,
AxEXGAUGELib._IGaugeEvents_TimerEvent e)
{
    EXGAUGELib.Layer var_Layer = axGauge1.Layers["rotateOnTimer"];
    var_Layer.RotateAngle = axGauge1.FormatABC("A +
5",.Layers["rotateOnTimer"].RotateAngle,null,null);
    System.Diagnostics.Debug.Print( e.tickCount.ToString() );
}
//this.axGauge1.Timer += new
AxEXGAUGELib._IGaugeEvents_TimerEventHandler(this.axGauge1_Timer);

axGauge1.set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotate

axGauge1.PicturesPath = "C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob";
axGauge1.PicturesName = "`Layer` + int(value + 1) + `.png`";
axGauge1.Layers.Count = 11;
axGauge1.Layers[0].Key = "rotateOnTimer";
axGauge1.TimerInterval = 1000;
```

## X++ (Dynamics Ax 2009)

```
// Timer event - Occurs when the interval elapses.
void onEvent_Timer(int _TickCount)
{
    COM com_Layer;
    anytype var_Layer;
    ;
    var_Layer = COM::createFromObject(exgauge1.Layers().Item("rotateOnTimer"));
    com_Layer = var_Layer;
    com_Layer.RotateAngle(exgauge1.FormatABC("A +
```



```

5",Layers().Item("rotateOnTimer").RotateAngle());
    print( _TickCount );
}

public void init()
{
    COM com_Layer;
    anytype var_Layer;
    ;

    super();

    exgauge1.DefaultLayer(185/*exDefLayerRotateType*/,COMVariant::createFromInt(1));
    exgauge1.PicturesPath("C:\\Program
Files\\Exontrol\\ExGauge\\Sample\\Design\\Circular\\Knob");
    exgauge1.PicturesName("`Layer` + int(value + 1) + `.png`");
    exgauge1.Layers().Count(11);
    var_Layer =
    COM::createFromObject(exgauge1.Layers()).Item(COMVariant::createFromInt(0));
    com_Layer = var_Layer;
    com_Layer.Key("rotateOnTimer");
    exgauge1.TimerInterval(1000);
}

```

## Delphi 8 (.NET only)

```

// Timer event - Occurs when the interval elapses.
procedure TWinForm1.AxGauge1_Timer(sender: System.Object; e:
AxEXGAUGELib._IGaugeEvents_TimerEvent);
begin
    with AxGauge1 do
    begin
        with Layers.Item['rotateOnTimer'] do
        begin
            RotateAngle := .FormatABC('A +
5',TObject(AxGauge1.Layers.Item['rotateOnTimer'].RotateAngle),Nil,Nil);

```

```

        OutputDebugString( e.tickCount );
    end;
end
end;

with AxGauge1 do
begin

set_DefaultLayer(EXGAUGELib.DefaultLayerPropertyEnum.exDefLayerRotateType,TObje

    PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
    PicturesName := 'Layer` + int(value + 1) + `.png`;
    Layers.Count := 11;
    Layers.Item[TObject(0)].Key := 'rotateOnTimer';
    TimerInterval := 1000;
end

```

## Delphi (standard)

```

// Timer event - Occurs when the interval elapses.
procedure TForm1.Gauge1Timer(ASender: TObject; TickCount : Integer);
begin
    with Gauge1 do
    begin
        with Layers.Item['rotateOnTimer'] do
        begin
            RotateAngle := .FormatABC('A +
5',OleVariant(Gauge1.Layers.Item['rotateOnTimer'].RotateAngle),Null,Null);
            OutputDebugString( TickCount );
        end;
    end
end;

with Gauge1 do
begin
    DefaultLayer[EXGAUGELib_TLB.exDefLayerRotateType] := OleVariant(1);

```

```

PicturesPath := 'C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob';
PicturesName := ``Layer` + int(value + 1) + `.png`;
Layers.Count := 11;
Layers.Item[OleVariant(0)].Key := 'rotateOnTimer';
TimerInterval := 1000;
end

```

## VFP

```

*** Timer event - Occurs when the interval elapses. ***
LPARAMETERS TickCount
with thisform.Gauge1
    with .Layers.Item("rotateOnTimer")
        .RotateAngle = thisform.Gauge1.FormatABC("A +
5",.Layers.Item("rotateOnTimer").RotateAngle)
        DEBUGOUT( TickCount )
    endwith
endwith

with thisform.Gauge1
    .Object.DefaultLayer(185) = 1
    .PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
    .PicturesName = ``Layer` + int(value + 1) + `.png`
    .Layers.Count = 11
    .Layers.Item(0).Key = "rotateOnTimer"
    .TimerInterval = 1000
endwith

```

## dBASE Plus

```

/*
with (this.EXGAUGEACTIVEXCONTROL1.nativeObject)
    Timer = class::nativeObject_Timer
endwith
*/
// Occurs when the interval elapses.

```

```

function nativeObject_Timer(TickCount)
    local var_Layer
    oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
    var_Layer = oGauge.Layers.Item("rotateOnTimer")
    var_Layer.RotateAngle = oGauge.FormatABC("A +
5",Layers.Item("rotateOnTimer").RotateAngle)
    ? Str(TickCount)
return

local oGauge

oGauge = form.EXGAUGEACTIVEXCONTROL1.nativeObject
oGauge.Template = [DefaultLayer(185) = 1] // oGauge.DefaultLayer(185) = 1
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.Layers.Item(0).Key = "rotateOnTimer"
oGauge.TimerInterval = 1000

```

## XBasic (Alpha Five)

```

' Occurs when the interval elapses.
function Timer as v (TickCount as N)
    Dim var_Layer as P
    oGauge = topparent:CONTROL_ACTIVEX1.activex
    var_Layer = oGauge.Layers.Item("rotateOnTimer")
    var_Layer.RotateAngle = oGauge.FormatABC("A +
5",Layers.Item("rotateOnTimer").RotateAngle)
    ? TickCount
end function

Dim oGauge as P

oGauge = topparent:CONTROL_ACTIVEX1.activex
oGauge.Template = "DefaultLayer(185) = 1" // oGauge.DefaultLayer(185) = 1

```

```

oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.Layers.Item(0).Key = "rotateOnTimer"
oGauge.TimerInterval = 1000

```

## Visual Objects

```

METHOD OCX_Exontrol1Timer(TickCount) CLASS MainDialog
    // Timer event - Occurs when the interval elapses.
    local var_Layer as ILayer
    var_Layer := oDCOCX_Exontrol1:Layers:[Item,"rotateOnTimer"]
        var_Layer:RotateAngle := oDCOCX_Exontrol1:FormatABC("A + 5";Layers:
[Item,"rotateOnTimer"]:RotateAngle,nil,nil)
        OutputDebugString(String2Psz( AsString(TickCount) ))
RETURN NIL

```

```

oDCOCX_Exontrol1:[DefaultLayer,exDefLayerRotateType] := 1
oDCOCX_Exontrol1:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oDCOCX_Exontrol1:PicturesName := "`Layer` + int(value + 1) + `.png`"
oDCOCX_Exontrol1:Layers.Count := 11
oDCOCX_Exontrol1:Layers:[Item,0]:Key := "rotateOnTimer"
oDCOCX_Exontrol1:TimerInterval := 1000

```

## PowerBuilder

```

/*begin event Timer(long TickCount) - Occurs when the interval elapses.*/
/*
    OleObject var_Layer
    oGauge = ole_1.Object
    var_Layer = oGauge.Layers.Item("rotateOnTimer")
        var_Layer.RotateAngle = oGauge.FormatABC("A +
5",Layers.Item("rotateOnTimer").RotateAngle)


```

```

        MessageBox("Information",string( String(TickCount) ))
    */
/*end event Timer*/

OleObject oGauge

oGauge = ole_1.Object
oGauge.DefaultLayer(185,1)
oGauge.PicturesPath = "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge.PicturesName = "`Layer` + int(value + 1) + `.png`"
oGauge.Layers.Count = 11
oGauge.Layers.Item(0).Key = "rotateOnTimer"
oGauge.TimerInterval = 1000

```

## Visual DataFlex

```

// Occurs when the interval elapses.
Procedure OnComTimer Integer  IITickCount
    Forward Send OnComTimer IITickCount
    Variant voLayers
    Get ComLayers to voLayers
    Handle hoLayers
    Get Create (RefClass(cComLayers)) to hoLayers
    Set pvComObject of hoLayers to voLayers
    Variant voLayer
    Get ComItem of hoLayers "rotateOnTimer" to voLayer
    Handle hoLayer
    Get Create (RefClass(cComLayer)) to hoLayer
    Set pvComObject of hoLayer to voLayer
    Variant v
        Variant vA
        Variant voLayers1
        Get ComLayers to voLayers1
        Handle hoLayers1
        Get Create (RefClass(cComLayers)) to hoLayers1

```

```
Set pvComObject of hoLayers1 to voLayers1
Variant voLayer1
Get ComItem of hoLayers1 "rotateOnTimer" to voLayer1
Handle hoLayer1
Get Create (RefClass(cComLayer)) to hoLayer1
Set pvComObject of hoLayer1 to voLayer1
    Get ComRotateAngle of hoLayer1 to vA
    Send Destroy to hoLayer1
Send Destroy to hoLayers1
Get ComFormatABC "A + 5" vA Nothing Nothing to v
Set ComRotateAngle of hoLayer to v
ShowIn IITickCount
Send Destroy to hoLayer
Send Destroy to hoLayers
End_Procedure
```

#### Procedure OnCreate

```
Forward Send OnCreate
Set ComDefaultLayer OLExDefLayerRotateType to 1
Set ComPicturesPath to "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
Set ComPicturesName to "`Layer` + int(value + 1) + `.png`"
Variant voLayers2
Get ComLayers to voLayers2
Handle hoLayers2
Get Create (RefClass(cComLayers)) to hoLayers2
Set pvComObject of hoLayers2 to voLayers2
    Set ComCount of hoLayers2 to 11
Send Destroy to hoLayers2
Variant voLayers3
Get ComLayers to voLayers3
Handle hoLayers3
Get Create (RefClass(cComLayers)) to hoLayers3
Set pvComObject of hoLayers3 to voLayers3
    Variant voLayer2
    Get ComItem of hoLayers3 0 to voLayer2
    Handle hoLayer2
```

```

    Get Create (RefClass(cComLayer)) to hoLayer2
    Set pvComObject of hoLayer2 to voLayer2
        Set ComKey of hoLayer2 to "rotateOnTimer"
    Send Destroy to hoLayer2
Send Destroy to hoLayers3
Set ComTimerInterval to 1000
End_Procedure

```

## XBase++

```

PROCEDURE OnTimer(oGauge,TickCount)
    LOCAL oLayer
    oLayer := oGauge:Layers:Item("rotateOnTimer")
        oLayer:RotateAngle := oGauge:FormatABC("A +
5",;Layers:Item("rotateOnTimer"):RotateAngle())
        DevOut( Transform(TickCount,"") )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oGauge

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oGauge := XbpActiveXControl():new( oForm:drawingArea )
    oGauge:CLSID := "Exontrol.Gauge.1" /*{91628F12-393C-44EF-A558-
83ED1790AAD3}*/
    oGauge:create(,, {10,60},{610,370} )

    oGauge:Timer := {|| TickCount| OnTimer(oGauge,TickCount)} /*Occurs when the

```



*interval elapses.\**

```
oGauge:SetProperty("DefaultLayer",185/*exDefLayerRotateType*/,1)
oGauge:PicturesPath := "C:\Program
Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
oGauge:PicturesName := "`Layer` + int(value + 1) + `.png`"
oGauge:Layers():Count := 11
oGauge:Layers:Item(0):Key := "rotateOnTimer"
oGauge:TimerInterval := 1000

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

# property Gauge.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

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

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

So, calling the TemplateDef property should be as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

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

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

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

# property Gauge.Template as String

Specifies the control's template.

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

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

The Exontrol's [eXHelper](#) tool helps you to find easy and quickly the answers and the source code for your questions regarding the usage of our UI components.

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.*

For instance, the following script:

```
PicturesPath = "C:\Program Files\Exontrol\ExGauge\Sample\Design\Circular\Knob"
```

```
PicturesName = "`Layer` + str(value + 1) + `.png`"
```

```
Layers.Count = 10
```

generates:





# method Gauge.ExecuteTemplate (Template as String)

Executes a template and returns the result.

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

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

For instance, the following sample retrieves the control's background color:

```
Debug.Print Gauge1.ExecuteTemplate("BackColor")
```

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

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

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

The Template 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.*