

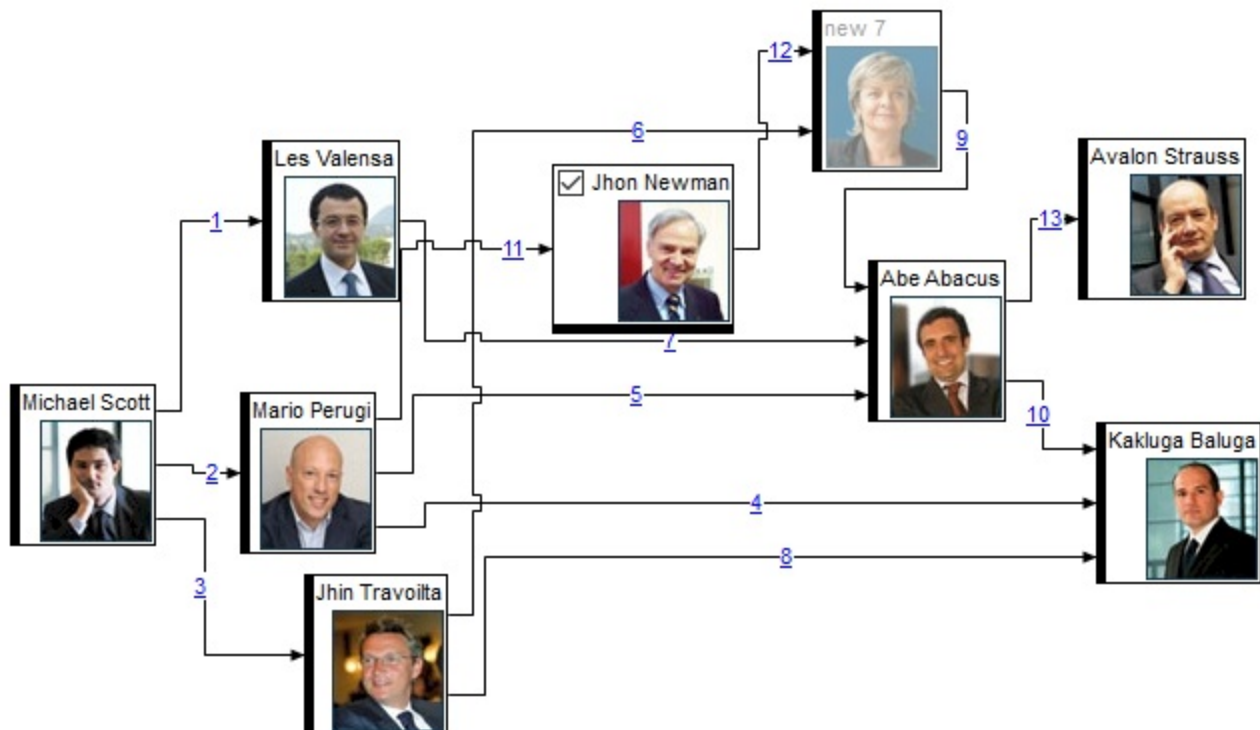


# ExSurface

The eXSurface lets you organize your objects to a surface. You can use the tool to generate organigrams, diagrams, graphs, flowcharts and so on. The ExSurface library lets the user changes its visual appearance using skins, each one providing an additional visual experience that enhances viewing pleasure.

Features include:

- Auto-Arrange support, or ability to automatically arrange horizontally or vertically the elements on the surface based on their relations, so they won't intersect one with another as much as possible
- Drag and Drop support
- Zooming support
- Undo/Redo support
- ActiveX controls hosting
- Tree-View support
- Cross/Junction/Intersection Links Support
- Skinnable Interface support ( ability to apply a skin to any background part )
- Unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the element's background.
- Print and Print - Preview support, Fit-To-Pages, ...
- Easy way to define the control's visual appearance in design mode, using XP-Theme elements or EBN objects



## How to get support?

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

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

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

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

Regards,  
Exontrol Development Team

<https://www.exontrol.com>

# constants AlignmentEnum

The AlignmentEnum type specifies the object's alignment. The AlignmentEnum type supports the following values:

Name	Value	Description
LeftAlignment	0	Left Alignment
CenterAlignment	1	Center Alignment
RightAlignment	2	Right Alignment
UpAlignment	3	Up Alignment
DownAlignment	4	Down Alignment

# constants AlignObjectsToGridLinesEnum

The AlignObjectsToGridLinesEnum type specifies how the objects are aligned to none, minor or major grid lines. The [AlignObjectsToGridLines](#) property specifies the way the objects are aligned to the control's grid lines. The AlignObjectsToGridLinesEnum type supports the following values:

Name	Value	Description
exAlignObjectsToNone	0	No alignment is performed.
exAlignObjectsToMinor	-1	The objects are aligned to minor grid lines.
exAlignObjectsToMajor	1	The objects are aligned to major grid lines.

# constants AllowKeysEnum

The AllowKeysEnum type specifies the keys to be combined in order to start an UI operation. For instance, the [AllowCreateObject](#) property of AllowKeysEnum type indicates the keys combination to let user creates a new element at runtime. By default, this property is set on exLeftClick + exDbClick, which means the user is able to create a new element by double left click. If this property is set on exRightClick + exCTRLKey the user should press the CTRL key while right clicking the control to start creating a new element. If the exDbClick flag is included, the user requires to do a double click instead single click to perform the operation. The exDisallow flag indicates that the operation is not allowed.

The AllowKeysEnum type supports the following values:

Name	Value	Description
exDisallow	0	exDisallow. The operation is not allowed.
exLeftClick	1	exLeftClick. The operation starts once the user clicks the left mouse button.
exRightClick	2	exRightClick. The operation starts if the user clicks the right mouse button.
exMiddleClick	3	exMiddleClick. The operation starts if the user clicks the middle mouse button.
exSHIFTKey	8	exSHIFTKey. The operation may start only if the user presses the SHIFT key.
exCTRLKey	16	exCTRLKey. The operation may start only if the user presses the CTRL key.
exALTKey	32	exALTKey. The operation may start only if the user presses the ALT key.
exDbClick	64	exDbClick. The operation starts only if the user double clicks, instead single click.

# constants AppearanceEnum

The AppearanceEnum enumeration is used to specify the appearance of the control's border.

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

# constants BackgroundExtPropertyEnum

The BackgroundExtPropertyEnum type specifies the UI properties of the part of the EBN you can access/change at runtime. The [BodyBackgroundExt](#) property specifies the EBN String format to be displayed on the element's background. The [BackgroundExtValue](#) property access the value of the giving property for specified part of the EBN. The BackgroundExtPropertyEnum type supports the following values:

Name	Value	Description
------	-------	-------------

Specifies the part's ToString representation. The [BodyBackgroundExt](#) property specifies the EBN String format to be displayed on the object's background. *The Exontrol's [eXButton](#) WYSWYG Builder helps you to generate or view the EBN String Format, in the **To String** field.*

Sample:

```
"client(right[18]  
(bottom[18,pattern=6,frame=0,framethick]),bottom[4  
(bottom[18,pattern=6,frame=0,framethick])"
```

generates the following layout:

exToStringExt

0

To String: client(right[18](bottom[18,pattern=0x006,frame=RGB(0,0,0),framethick]),bo

Root

Client

Right:s

Bottom:s

Bottom:s

Left:s

Bottom:s

where it is applied to an object it looks as follows:



(String expression, read-only).

exBackColorExt

1

Indicates the background color / EBN color to be shown on the part of the object. *Sample: 255 indicates red, RGB(0,255,0) green, or 0x1000000.*

*(Color/Numeric expression, The last 7 bits in the high significant byte of the color indicate the identifier of the skin being used )*

Specifies the position/size of the object, depending on the object's anchor. The syntax of the exClientExt is related to the exAnchorExt value. *For instance, if the object is anchored to the left side of the parent ( exAnchorExt = 1 ), the exClientExt specifies just the width of the part in pixels/percents, not including the position. In case, the exAnchorExt is client, the exClientExt has no effect.*

*Based on the exAnchorExt value the exClientExt is:*

- *0 (**none**, the object is not anchored to any side), the format of the exClientExt is "**left,top,width,height**" ( as string ) where (left,top) margin indicates the position where the part starts, and the (width,height) pair specifies its size. The left, top, width or height could be any expression (+,-,/ or \* ) that can include numbers associated with pixels or percents. For instance: "25%,25%,50%,50%" indicates the middle of the parent object, and so when the parent is resized the client is resized accordingly. The "50%-8,50%-8,16,16" value specifies that the size of the object is always 16x16 pixels and positioned on the center of the parent object.*
- *1 (**left**, the object is anchored to left side of the parent), the format of the exClientExt is **width** ( string or numeric ) where width indicates the width of the object in pixels, percents or a combination of them using +,-,/ or \* operators. For instance: "50%" indicates*



exClientExt

2

- the half of the parent object, and so when the parent is resized the client is resized accordingly. The 16 value specifies that the size of the object is always 16 pixels.*
- 2 (**right**, the object is anchored to right side of the parent object), the format of the exClientExt is **width** ( string or numeric ) where width indicates the width of the object in pixels, percents or a combination of them using +,-,/ or \* operators. For instance: "50%" indicates the half of the parent object, and so when the parent is resized the client is resized accordingly. The 16 value specifies that the size of the object is always 16 pixels.
  - 3 (**client**, the object takes the full available area of the parent), the exClientExt has no effect.
  - 4 (**top**, the object is anchored to the top side of the parent object), the format of the exClientExt is **height** ( string or numeric ) where height indicates the height of the object in pixels, percents or a combination of them using +,-,/ or \* operators. For instance: "50%" indicates the half of the parent object, and so when the parent is resized the client is resized accordingly. The 16 value specifies that the size of the object is always 16 pixels.
  - 5 (**bottom**, the object is anchored to bottom side of the parent object), the format of the exClientExt is **height** ( string or numeric ) where height indicates the height of the object in pixels, percents or a combination of them using +,-,/ or \* operators. For instance: "50%" indicates the half of the parent object, and so when the parent is resized the client is resized accordingly. The 16 value specifies that the size of the object is always 16 pixels.

*Sample: 50% indicates half of the parent, 25 indicates 25 pixels, or 50%-8 indicates 8-pixels left from the center of the parent.*

*(String/Numeric expression)*

exAnchorExt

3

Specifies the object's alignment relative to its parent.

*The valid values for exAnchorExt are:*

- *0 (**none**), the object is not anchored to any side,*
- *1 (**left**), the object is anchored to left side of the parent,*
- *2 (**right**), the object is anchored to right side of the parent object,*
- *3 (**client**), the object takes the full available area of the parent,*
- *4 (**top**), the object is anchored to the top side of the parent object,*
- *5 (**bottom**), the object is anchored to bottom side of the parent object*

*(Numeric expression)*

Specifies the HTML text to be displayed on the object.

*The exTextExt supports the following built-in HTML tags:*

- **<b> ... </b>** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... </a>** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The

*FormatAnchor* property customizes the visual effect for anchor elements.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using `<a ;exp=>` or `<a ;e64=>` anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAA</a>`" that displays `show lines-` in gray when the user clicks the + anchor. The "`gA8ABmABnABjABvABshIAOQAEAAHAA`" string encodes the "`<fgcolor 808080>show lines<a>-</a></fgcolor>`"  
The `Decode64Text/Encode64Text` methods of the `eXPrint` can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "`<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3`" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **`<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&gt;**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **<font face;size>** to define a smaller or a larger font to be displayed. For instance: "Text with **<font ;7><off 6>**subscript" displays the text such as: Text with subscript  
The "Text with **<font ;7><off -6>**superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the

rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `<font>` HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "`<font ;18><gra FFFFFFFF;1;1>gradient-center</gra></font>`" generates the following picture:

gradient-center

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

outlined

- `<sha rrggbb;width;offset> ... </sha>` define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `<font>` HTML tag can be used to define the height of the font. For instance the "`<font ;31><sha>shadow</sha></font>`" generates the following picture:

shadow

or "`<font ;31><sha 404040;5;0>`

`<fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha></font>`" gets:

outline anti-aliasing

*(String expression)*

exTextExtWordWrap

5

Specifies that the object is wrapping the text. The exTextExt value specifies the HTML text to be displayed on the part of the EBN object. This property has effect only if there is a text assigned to the part using the exTextExt flag.

*(Boolean expression)*

Indicates the alignment of the text on the object. The exTextExt value specifies the HTML text to be displayed on the part of the EBN object. This property has effect only if there is a text assigned to the part using the exTextExt flag.

*The valid values for exTextExtAlignment are:*

- 0, ( hexa 0x00, **Top-Left** ), Text is vertically aligned at the top, and horizontally aligned on the left.
- 1, ( hexa 0x01, **Top-Center** ), Text is vertically aligned at the top, and horizontally aligned at the center.
- 2, ( hexa 0x02, **Top-Right** ), Text is vertically aligned at the top, and horizontally aligned on the right.
- 16, ( hexa 0x10, **Middle-Left** ), Text is vertically aligned in the middle, and horizontally aligned on the left.
- 17, ( hexa 0x11, **Middle-Center** ), Text is vertically aligned in the middle, and horizontally aligned at the center.
- 18, ( hexa 0x12, **Middle-Right** ), Text is vertically aligned in the middle, and horizontally aligned on the right.

exTextExtAlignment






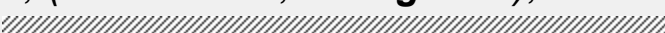
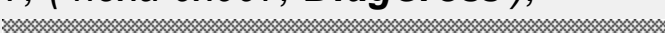




6

- 32, ( hexa 0x20, **Bottom-Left** ), Text is vertically aligned at the bottom, and horizontally aligned on the left.
- 33, ( hexa 0x21, **Bottom-Center** ), Text is vertically aligned at the bottom, and horizontally aligned at the center.
- 34, ( hexa 0x22, **Bottom-Right** ), Text is vertically aligned at the bottom, and horizontally aligned on the right.

(Numeric expression)

Indicates the pattern to be shown on the object. The exPatternColorExt specifies the color to show the pattern.




The valid values for exPatternExt are:

- 0, ( hexa 0x000, **Empty** ), The pattern is not visible
- 1, ( hexa 0x001, **Solid** ),  

- 2, ( hexa 0x002, **Dot** ),  

- 3, ( hexa 0x003, **Shadow** ),  

- 4, ( hexa 0x004, **NDot** ),  

- 5, ( hexa 0x005, **FDiagonal** ),  

- 6, ( hexa 0x006, **BDiagonal** ),  

- 7, ( hexa 0x007, **DiagCross** ),  

- 8, ( hexa 0x008, **Vertical** ),  

- 9, ( hexa 0x009, **Horizontal** ),  

- 10, ( hexa 0x00A, **Cross** ),  

- 11, ( hexa 0x00B, **Brick** ),  


exPatternExt

7



- 12, ( *hexa 0x00C*, **Yard** ),  

- 256, ( *hexa 0x100*, **Frame** ),  
. The *exFrameColorExt* specifies the color to show the frame. The **Frame** flag can be combined with any other flags.
- 768, ( *hexa 0x300*, **FrameThick** ),  
. The *exFrameColorExt* specifies the color to show the frame. The **Frame** flag can be combined with any other flags.

*(Numeric expression)*

*exPatternColorExt*

8

Indicates the color to show the pattern on the object. The *exPatternColorExt* property has effect only if the *exPatternExt* property is not 0 ( empty ). The *exFrameColorExt* specifies the color to show the frame ( the *exPatternExt* property includes the **exFrame** or **exFrameThick** flag )

*(Color expression)*

*exFrameColorExt*

9

Indicates the color to show the border-frame on the object. This property set the **Frame** flag for *exPatternExt* property.

*(Color expression)*

*exFrameThickExt*

10

Specifies that a thick-frame is shown around the object. This property set the **FrameThick** flag for *exPatternExt* property.

*(Boolean expression)*

*exUserDataExt*

11

Specifies an extra-data associated with the object.  
*(Variant expression)*

# 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	Indicates 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. The <a href="#">ToolTipDelay</a> property specifies the time in ms that passes before the ToolTip appears. Use the <a href="#">ToolTip</a> property of the Element object to specify the element's tooltip. Use the <a href="#">ToolTipWidth</a> property to specify the width of the tooltip window. Use the <a href="#">ToolTip</a> property of the Link object to specify the tooltip to be shown when the cursor hovers the link. 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.
exCheckBoxState0	70	Specifies the visual appearance for the check box in 0 state (unchecked). Use the <a href="#">ShowCheckBox</a> property to show or hide the element's checkbox. Use the <a href="#">Checked</a> property to specify the state of the element's checkbox.
exCheckBoxState1	71	Specifies the visual appearance for the check box in 1 state (checked). Use the <a href="#">ShowCheckBox</a> property to show or hide the element's checkbox. Use the <a href="#">Checked</a> property to specify the state of the element's checkbox.
exCheckBoxState2	72	Specifies the visual appearance for the check box in 2 state (partial, reserved). Use the <a href="#">ShowCheckBox</a> property to show or hide the element's checkbox. Use the <a href="#">Checked</a> property to specify the state of the element's checkbox.

exRadioButtonState0	73	(reserved) Specifies the visual appearance for the radio button in 0 state (unchecked).
exRadioButtonState1	74	(reserved) Specifies the visual appearance for the radio button in 1 state (checked).
exCreateObjectColor	75	Specifies the color to show the creation rectangle when the user creates a new object. The <a href="#">AllowCreateObject</a> property specifies the combination of keys that allows the user to create objects on the surface.
exSelectObjectRectColor	76	Specifies the color to show the selection rectangle when the objects are selected by dragging. The <a href="#">AllowSelectObjectRect</a> property specifies the keys combination so the user can select the elements from the dragging rectangle. The <a href="#">SelectObjectColor</a> / <a href="#">SelectObjectTextColor</a> property specifies the colors to show the selected elements ( while the control has the focus ). The <a href="#">SelectObjectColorInactive</a> / <a href="#">SelectObjectTextColorInactive</a> property specifies the color to show the selected elements ( while the control is not focused ). The <a href="#">SelectObjectStyle</a> property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ).
exElementHostWindowBackColor	77	Specifies the visual appearance of the element of window/control type while dragging.
exEditBackColor	78	Specifies the background color while element is editing. Use the <a href="#">Edit</a> method to allow inline editing of the element's caption or extracaption.
exEditForeColor	79	Specifies the foreground color while element is editing. Use the <a href="#">Edit</a> method to allow inline editing of the element's caption or extracaption.
exEditSelBackColor	80	Specifies the selection background color while element is editing. Use the <a href="#">Edit</a> method to allow inline editing of the element's caption or extracaption.
exEditSelForeColor	81	Specifies the selection foreground color while element is editing. Use the <a href="#">Edit</a> method to allow inline editing of the element's caption or

extracaption.

exLinkObjectsInvalidColor	82	Specifies the color to display an invalid link, when the user links two objects. The <a href="#">AllowLink</a> event notifies your application that the user links two elements on the surface.
exLinkObjectsValidColor	83	Specifies the color to display a valid link, when the user links two objects. The <a href="#">AllowLink</a> event notifies your application that the user links two elements on the surface.
exTreeGlyphCollapsed	84	Specifies the visual appearance to show the glyph next to the collapsed element. The <a href="#">Expanded</a> property specifies whether the element is expanded or collapsed. The <a href="#">ExpadedLinkedElements</a> property specifies whether the element that has outgoing links displays the +/- expanding button.
exTreeGlyphExpanded	85	Specifies the visual appearance to show the glyph next to the expanded element. The <a href="#">Expanded</a> property specifies whether the element is expanded or collapsed. The <a href="#">ExpadedLinkedElements</a> property specifies whether the element that has outgoing links displays the +/- expanding button.
exHoverInsertObject	86	Specifies the visual appearance to display the border of the element where the dragging object is about to be inserted.
exHoverInsertObjectGlyph	87	Specifies the visual appearance of the glyph where the dragging object is about to be inserted.
exElementBorderColor	88	Specifies the color or the visual appearance of the element's border. If the property is set on -1, no border is shown. The <a href="#">BorderColor</a> property specifies the color to show the border for a specific element. The <a href="#">BorderPadding</a> property specifies the border padding.
exElementStatusColor	89	Specifies the color or the visual appearance of the element's status. If the property is set on -1, no status is shown. Still, the element's <a href="#">StatusSize</a> should be set on 0. The <a href="#">StatusColor</a> property specifies the color or the visual appearance to show the element's status part. The <a href="#">StatusPattern</a> property specifies the pattern to show the element's status part. The <a href="#">StatusAlign</a> property indicates the

alignment of the element's status.

exElementBackColor	90	Specifies the color or the visual appearance of the element's background. If -1, the element's background is transparent. The <a href="#">BackColor</a> property defines the element's background color or visual appearance.
exElementForeColor	91	Specifies the element's foreground color. The <a href="#">ForeColor</a> property defines the element's foreground color or visual appearance.
exContextMenuAppearance	99	Specifies the visual appearance of the control's context menu.
exContextMenuBackColor	100	Specifies the solid background color for the control's context menu.
exContextMenuForeColor	101	Specifies the text foreground color for the control's context menu.
exContextMenuSelBackColor	102	Specifies the solid/EBN selection's background color in the control's context menu.
exContextMenuSelBorderColor	103	Specifies the solid color to show the selection in the control's context menu.
exContextMenuSelForeColor	104	Specifies the selection's text foreground color in the control's context menu.
exToolBarAppearance	148	exToolBarAppearance. Specifies the visual appearance of the surface's toolbar panel.
exToolBarBackColor	149	exToolBarBackColor. Specifies the background color of the surface's toolbar panel.
exToolBarForeColor	150	exToolBarForeColor. Specifies the foreground color of the surface's toolbar panel.
exToolBarButtonUpBackColor	151	exToolBarButtonUpBackColor. Specifies the visual appearance of the toolbar's button while it is up.
exToolBarButtonUpForeColor	152	exToolBarButtonUpForeColor. Specifies the foreground color of the toolbar's button while it is up.
exToolBarButtonDownBackColor	153	exToolBarButtonDownBackColor. Specifies the visual appearance of the toolbar's button while it is down.
exToolBarButtonDownForeColor	154	exToolBarButtonDownForeColor. Specifies the foreground color of the toolbar's button while it is

down.

exToolBarButtonHotBackColor 155

exToolBarButtonHotBackColor. Specifies the visual appearance of the toolbar's button while the cursor hovers it.

exToolBarButtonHotForeColor 156

exToolBarButtonHotForeColor. Specifies the foreground color of the toolbar's button while the cursor hovers it.

# constants CaptionSingleLineEnum

The CaptionSingleLineEnum type defines whether the element's caption is displayed on a single or multiple lines. The ([ExtraCaptionSingleLine/](#))[CaptionSingleLine](#) property retrieves or sets a value indicating whether the element's (extra/)caption is displayed using one line, or multiple lines. The CaptionSingleLineEnum type supports the following values:

Name	Value	Description
exCaptionSingleLine	-1	<p>Indicates that the element's caption is displayed on a single line. In this case any \r\n or &lt;br&gt; HTML tags is ignored. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as:</p> <div>This is the fir...</div>
exCaptionWordWrap	0	<p>Specifies that the element's caption is displayed on multiple lines, by wrapping the words. Any \r\n or &lt;br&gt; HTML tag breaks the line. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as:</p> <div>This is the first line. This is the second line. This is the third line.</div>
exCaptionBreakWrap	1	<p>Specifies that the element's caption is displayed on multiple lines, by wrapping the breaks only. Only The \r\n or &lt;br&gt; HTML tag breaks the line. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as:</p> <div>This is the fir... This is the se... This is the thi...</div>

# constants CheckStateEnum

The CheckStateEnum type specifies the states of the element's check-box. The [Checked](#) property specifies the state of the element's check box. Use the [ShowCheckBox](#) property to show or hide the element's checkbox. The CheckStateEnum type supports the following values:

Name	Value	Description
exUnchecked	0	Specifies whether the element is unchecked. The <a href="#">Background</a> (exCheckBoxState0) Specifies the visual appearance for the check box in 0 state.
exChecked	1	Specifies whether the element is checked. The <a href="#">Background</a> (exCheckBoxState1) Specifies the visual appearance for the check box in 1 state.
exPartialChecked	2	(reserved) Specifies whether the element is partial-checked. The <a href="#">Background</a> (exCheckBoxState2) Specifies the visual appearance for the check box in 2 state.



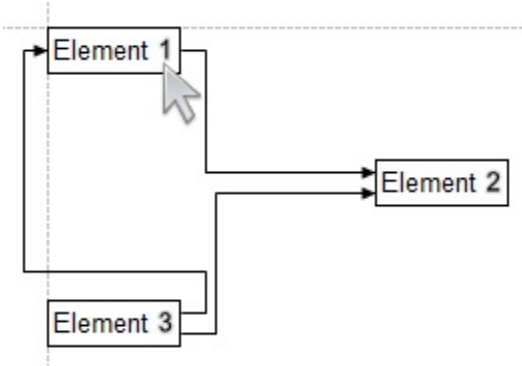
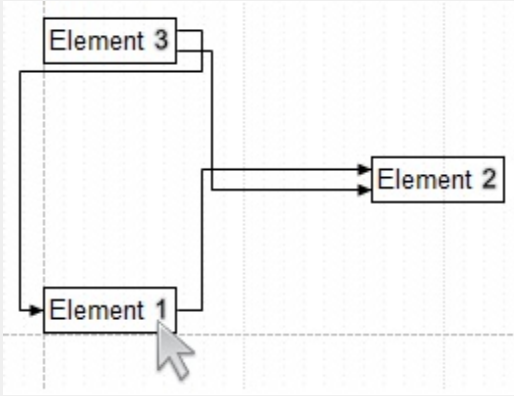
# constants ContentAlignmentEnum

The ContentAlignmentEnum type specifies the object's alignment relative to the corners. The ContentAlignmentEnum type supports the following values:

Name	Value	Description
exTopLeft	0	Content is vertically aligned at the top, and horizontally aligned on the left.
exTopCenter	1	Content is vertically aligned at the top, and horizontally aligned at the center.
exTopRight	2	Content is vertically aligned at the top, and horizontally aligned on the right.
exMiddleLeft	16	Content is vertically aligned in the middle, and horizontally aligned on the left.
exMiddleCenter	17	Content is vertically aligned in the middle, and horizontally aligned at the center.
exMiddleRight	18	Content is vertically aligned in the middle, and horizontally aligned on the right.
exBottomLeft	32	Content is vertically aligned at the bottom, and horizontally aligned on the left.
exBottomCenter	33	Content is vertically aligned at the bottom, and horizontally aligned at the center.
exBottomRight	34	Content is vertically aligned at the bottom, and horizontally aligned on the right.

# constants CoordEnum

The CoordEnum type defines the type of coordinates the elements of the surface supports. The [Coord](#) property specifies the type of coordinates the elements of the surface display in. The CoordEnum type supports the following values:

Name	Value	Description
exDefCoord	0	<p>The positive coordinates are shown right-down to origin of the surface. The following screen shot shows the surface using default coordinates:</p> 
exCartesian	1	<p>The elements show in Cartesian coordinates. The positive coordinates are shown right-up to origin of the surface. The following screen shot shows the surface using Cartesian coordinates:</p> 
exAllowPositiveOnly	16	<p>Only the positive panel of the surface is shown. The exAllowPositiveOnly flag can be combined with exDefCoord or exCartesian value. For instance, the exCartesian + exAllowPositiveOnly indicates that surface displays only the positive coordinates in Cartesian system.</p>

# constants DefArrangeEnum

The DefArrangeEnum type specifies the options of the [Arrange](#) method. The [DefArrange](#) property specifies a property of [Arrange](#) method. Changing any of the following properties has effect at the next Arrange call. The DefArrangeEnum type supports the following values:

Name	Value	Description
exDefArrangeDir	0	<p>Specifies the direction to auto-arrange the elements. By default, the exDefArrangeDir is 0, which indicates that the elements are horizontally arranged. The exDefArrangeDir property could be any of the following:</p> <ul style="list-style-type: none"><li>• 0, horizontal arrangement</li><li>• any other value, specifies a vertical arrangement.</li></ul> <p>(long expression)</p>
exDefArrangeDX	1	<p>Specifies the distance between two auto-arranged elements on horizontal axis. By default, the exDefArrangeDX property is 24 pixels. You can use the exDefArrangeDX to increase or decrease the distance of the arranged elements.</p> <p>(long expression)</p>
exDefArrangeDY	2	<p>Specifies the distance between two auto-arranged elements on vertical axis. By default, the exDefArrangeDY property is 18 pixels. You can use the exDefArrangeDX to increase or decrease the distance of the arranged elements.</p> <p>(long expression)</p>
exDefArrangeAlign	3	<p>Specifies the alignment of the elements relative to incoming/outgoing elements during the Arrange operation. By default, the exDefArrangeAlign property is CenterAlignment, which indicates that the elements are centered relative to incoming/outgoing elements</p>

([AlignmentEnum](#) expression)

exDefArrangeCompact	4	exDefArrangeCompact. Specifies whether the elements should be compacted, during the Arrange operation.
---------------------	---	--------------------------------------------------------------------------------------------------------

# constants EdgeAlignmentEnum

The EdgeAlignmentEnum type specifies the alignment of the object to the edges of the parent. The EdgeAlignmentEnum type supports the following values:

Name	Value	Description
exAlignLeft	0	The object is aligned to the left edge.
exAlignRight	1	The object is aligned to the right edge.
exAlignTop	2	The object is aligned to the top edge.
exAlignBottom	3	The object is aligned to the bottom edge.

# constants EditEnum

The EditEnum type specifies the caption to be edited on the element. The Part parameter of the Edit method specifies the caption of the element to be edited. The EditEnum type supports the following values:

Name	Value	Description
exEditCaption	0	Edits the element's caption.
exEditExtraCaption	1	Edits the element's extra caption.

# constants ElementHostTypeEnum

The ElementHostTypeEnum type defines the type of nodes that you can put on the surface. The ElementHostTypeEnum type supports the following values:

Name	Value	Description
exElementHostDefault	0	Specifies the default element. Use the <a href="#">Caption</a> property to specify the element's caption.
exElementHostWindow	1	Specifies an element that hosts a window. Use the <a href="#">Window</a> property to associate an existent window with the current element.
exElementHostControl	2	Specifies an element that hosts a control. Use the <a href="#">Control/License</a> property to associate an inner control or an ActiveX control.

# constants exClipboardFormatEnum

Defines the clipboard format constants. Use [GetFormat](#) property to check whether the clipboard data is of given type

Name	Value	Description
exCFText	1	Null-terminated, plain ANSI text in a global memory bloc
exCFBitmap	2	A bitmap compatible with Windows 2.X
exCFMetafile	3	A Windows metafile with some additional information about how the metafile should be displayed
exCFDIB	8	A global memory block containing a Windows device-independent bitmap (DIB)
exCFPalette	9	A color-palette handle
exCFEMetafile	14	A Windows enhanced metafile
exCFFiles	15	A collection of files. Use <a href="#">Files</a> property to get the collection of files
exCFRTF	-16639	A RTF document



# constants exOLEDragOverEnum

State transition constants for the OLEDragOver event.

Name	Value	Description
exOLEDragEnter	0	Source component is being dragged within the range of a target.
exOLEDragLeave	1	Source component is being dragged out of the range of a target.
exOLEDragOver	2	Source component has moved from one position in the target to another.

# constants exOLEDropEffectEnum

Drop effect constants for OLE drag and drop events.

Name	Value	Description
exOLEDropEffectNone	0	Drop target cannot accept the data, or the drop operation was cancelled
exOLEDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
exOLEDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
exOLEDropEffectScroll	-2147483648	Not implemented.

# constants exOLEDropModeEnum

Constants for the OLEDropMode property, that defines how the control accepts OLE drag and drop operations. Use the [OLEDropMode](#) property to set how the component handles drop operations.

Name	Value	Description
exOLEDropNone	0	The control is not used OLE drag and drop functionality
exOLEDropManual	1	The control triggers the OLE drop events, allowing the programmer to handle the OLE drop operation in code

Here's the list of events related to OLE drag and drop: [OLECompleteDrag](#), [OLEDragDrop](#), [OLEDragOver](#), [OLEGiveFeedback](#), [OLESetData](#), [OLEStartDrag](#).

# constants HitTestCodeEnum

The HitTestCodeEnum type specifies the parts of the element being handled by the [HitTestFromPoint](#) property. The exHitTestClient... flags can be combined with any other flags. The HitTestCodeEnum type supports the following values:

Name	Value	Description
exHitTestInvalid	-1	Invalid hit-test code.
exHitTestMargin	0	Indicates a margin of the element.
exHitTestStatus	1	Indicates the status part of the element.
exHitTestClient	2	Indicates the client part of the element.
exHitTestPicture	3	Indicates a picture of the element.
exHitTestCaption	4	Indicates the caption of the element.
exHitTestExtraCaption	5	Indicates the extra caption of the element.
exHitTestCheckBox	6	Indicates the check-box of the element.
exHitTestGlyph	7	Indicates the glyph of the element.
exHitTestMask	255	Indicates the mask for the hit-test code.
exHitTestClientTopLeft	0	Indicates the top-left portion of the object.
exHitTestClientTopCenter	256	Indicates the top-center portion of the object.
exHitTestClientTopRight	512	Indicates the top-right portion of the object.
exHitTestClientMiddleLeft	4096	Indicates the middle-left portion of the object.
exHitTestClientMiddleCenter	4352	Indicates the midle-center portion of the object.
exHitTestClientMiddleRight	4608	Indicates the midle-right portion of the object.
exHitTestClientBottomLeft	8192	Indicates the bottom-left portion of the object.
exHitTestClientBottomCenter	8448	Indicates the bottom-center portion of the object.
exHitTestClientBottomRight	8704	Indicates the bottom-right portion of the object.

## constants LayoutChangingEnum

The LayoutChangingEnum type specifies the operations that the user performs on the surface. The [LayoutStartChanging](#) event occurs when a specified operation begins. The [LayoutEndChanging](#) event notifies that the specified operation ends. The LayoutChangingEnum type supports the following values.

Name	Value	Description
exSurfaceMove	0	The user moves/scrolls the surface to a new position. The <a href="#">AllowMoveSurface</a> property specifies the keys combination to allow user to move / scroll the surface.
exSurfaceZoom	1	The user magnifies or shrinks the surface (zooming). The <a href="#">AllowZoomSurface</a> property specifies the keys combination to allow user to zoom the surface.
exSurfaceHome	2	The user restores the surface to its original view. The <a href="#">Home</a> method restores the view to its original state ( position and zoom ).
exResizeObject	3	The user resizes an object on the surface.
exMoveObject	4	The user moves an object on the surface.
exSelectObject	5	The user selects objects on the surface.
exSelectNothing	6	The user selects nothing on the surface. The <a href="#">AllowSelectNothing</a> property specifies the keys combination to allow user to select nothing on the surface.
exCreateObject	7	The user creates objects on the surface.
exEditObject	8	The user edits the element's caption.
exLinkObjects	9	The user links elements. The <a href="#">AllowLinkObjects</a> property specifies the keys combination to allow user to link elements on the surface.
exLinkControlPoint	19	The user changes the link's control points.
exFocusLink	20	The user clicks a link (the focused link is being updated). The <a href="#">FocusLink</a> property retrieves or changes the current link that is currently focused (selected or active) within the control.
exUndo	33	An Undo operation is performed (CTR + Z). Occurs only if the control's <a href="#">AllowUndoRedo</a> property is

True.

exRedo

34

A Redo operation is performed (CTR + Y). Occurs only if the control's [AllowUndoRedo](#) property is True.

exUndoRedoUpdate

32

The Undo/Redo queue is updated.

# constants LinesStyleEnum

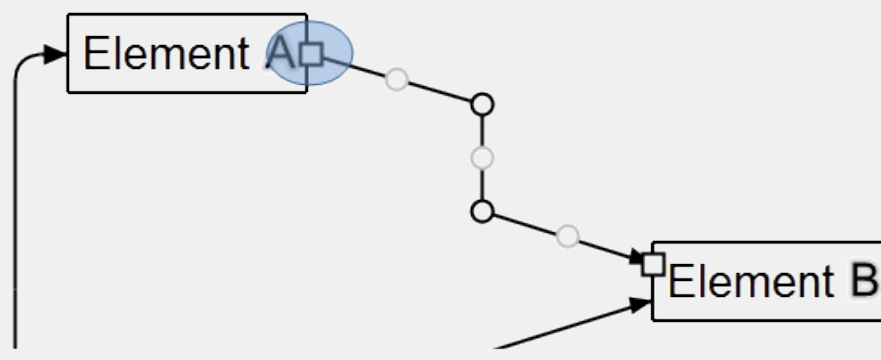
The LinesStyleEnum type specifies the type of lines the control can show. The LinesStyleEnum type supports the following values:

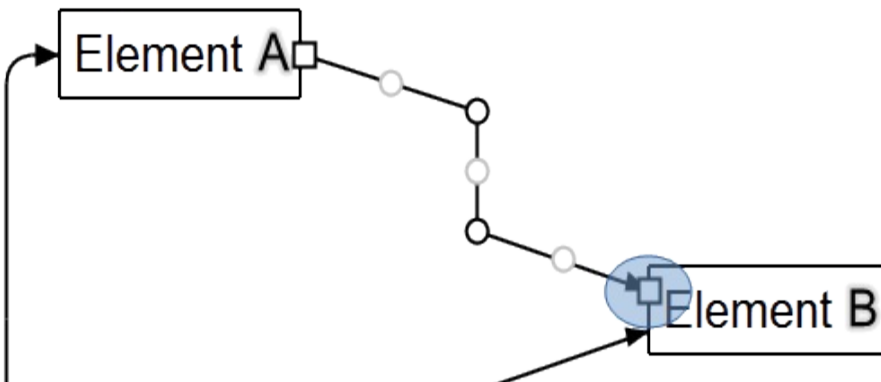
Name	Value	Description
exNoLines	-1	No lines are shown.
exLinesDot	0	The lines shows as dotted.
exLinesHDot4	1	The horizontal lines shows dotted.
exLinesVDot4	2	The vertical lines are shown as dotted.
exLinesDot4	3	The lines are shown as solid.
exLinesHDash	4	The horizontal lines are shown as dashed.
exLinesVDash	8	The vertical lines are shown as dashed.
exLinesDash	12	The lines are shown as dashed.
exLinesHSolid	16	The horizontal lines are shown as solid.
exLinesVSolid	32	The vertical lines are shown as solid.
exLinesSolid	48	The lines are shown as solid.
exLinesThick	256	The lines are shown ticker. This flag can be combined with any other flags, so the line is shown ticker.
exLinesThicker	768	The lines are shown ticker. This flag can be combined with any other flags, so the line is shown ticker.

# constants LinkControlPointEnum

The LinkControlPointEnum type specifies the link's control points. The LinkControlPointEnum type supports the following values:

Name	Value	Description
------	-------	-------------

exNoControlPoint	0	The link displays no control points.
exStartControlPoint	1	<p>The link shows control point that changes the link's <a href="#">StartPos</a> property. Can be combined with exAllowChangeFrom flag. The exStartControlPoint point is marked with black squares as shown in the following picture:</p> 

exEndControlPoint	2	<p>The link shows control point that changes the link's <a href="#">EndPos</a> property. Can be combined with exAllowChangeTo flag. The exEndControlPoint point is marked with black squares as shown in the following picture:</p> 
-------------------	---	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

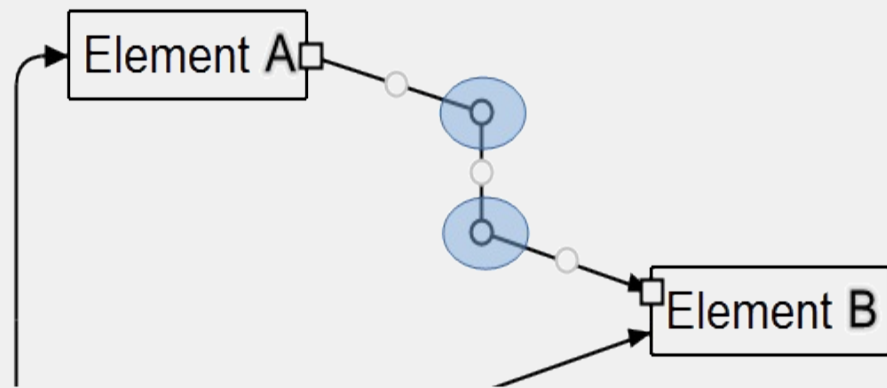
<p>Defines the corners of the link's path. You can remove a exControlPoint points by dragging to another, so intermediate exControlPoint points are removed. You can move all control points of the link</p>		
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	--



at once, if SHIFT key is pressed. The exControlPoint points are marked black circles as shown in the following picture:

exControlPoint

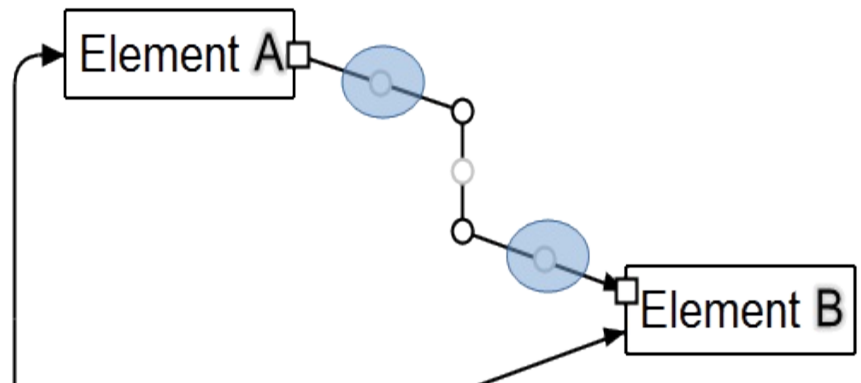
4



Defines the link's middle control points that are displayed between two exControlPoint points, to let the use add new exControlPoint points, to redefine the link's path. The exMiddleControlPoint points are marked with gray circles as shown in the following picture:

exMiddleControlPoint

8



exOrthoArrange

16

The exOrthoArrange flag specifies that the lines of the link are orthogonal arranged when the user drag and drop the middle or control-points of the path (excludes the start/end control-points). The flag has effect only for links that contains horizontal/vertical lines (orthogonal link). The cursor is changing to size-all when the mouse pointer hovers the control-points (excludes the start/end control-points), to size WE or NS when the mouse pointer hovers the control-points, else it is changed to hand cursor. The user can freely customize the link by keeping

the CTRL key down.

exAllowChangeFrom

32

The exAllowChangeFrom(0x20) flag allows the user to adjust the link's from element by dragging and dropping the start control point (requires the exStartControlPoint flag)

exAllowChangeTo

64

The exAllowChangeTo(0x40) flag indicates that the user can adjust the link's to element by dragging and dropping the end control point (requires the exEndControlPoint flag)

# constants LinkStyleEnum

The LinkStyleEnum type defines the style of lines to be shown on the surface. The LinkStyleEnum type supports the following values:

Name	Value	Description
exLinkSolid	0	The link is solid.
exLinkDash	1	The link is dashed.
exLinkDot	2	The link is dotted.
exLinkDashDot	3	The link has alternating dashes and dots.
exLinkDashDotDot	4	The link has alternating dashes and double dots.

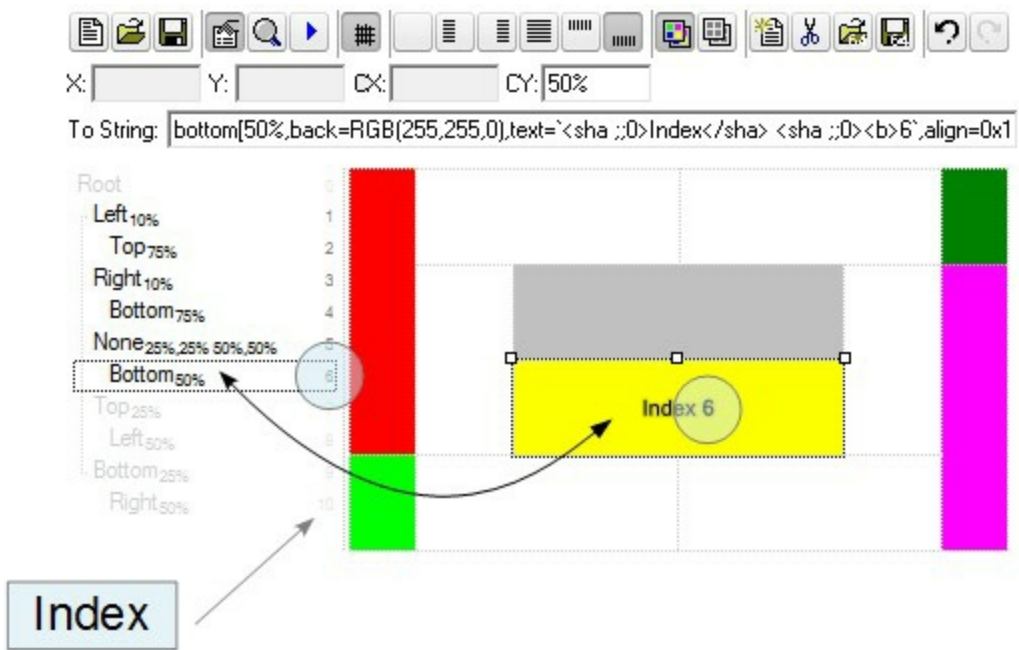
# constants MoveNeighborsEnum

The MoveNeighborsEnum type specifies how the elements are shifted once an element is moved. The The [AllowMoveNeighbors](#) property indicates whether the neighbor elements are shifted once the selection is moved or resized, so they won't intersect the dragging objects. The MoveNeighborsEnum type supports the following values:

Name	Value	Description
exDisallowMoveNeighbors	0	No element is moved.
exMoveNeighborsVertically	1	The neighbor elements of the element that's currently moving, are shifted vertically.
exMoveNeighborsHorizontally	2	The neighbor elements of the element that's currently moving, are shifted horizontally.
exMoveNeighborsByDragDir	3	The neighbor elements of the element that's currently moving, are shifted based on the direction of moving.

# constants IndexExtEnum

The IndexExtEnum type specifies the index of the part of the EBN object to be accessed. The Index parameter of the [BackgroundExtValue](#) property indicates the index of the part of the EBN object to be changed or accessed. *The Exontrol's [eXButton](#) WYSWYG Builder helps you to generate or view the EBN String Format, in the **To String** field.* The list of objects that compose the EBN are displayed on the left side of the Builder tool, and the Index of the part is displayed on each item aligned to the right as shown in the following screen shot:



In this sample, there are 11 objects that composes the EBN, so the Index property goes from 0 which indicates the root, and 10, which is the last item in the list

So, let's apply this format to an object, to change the exPatternExt property for the object with the Index 6:

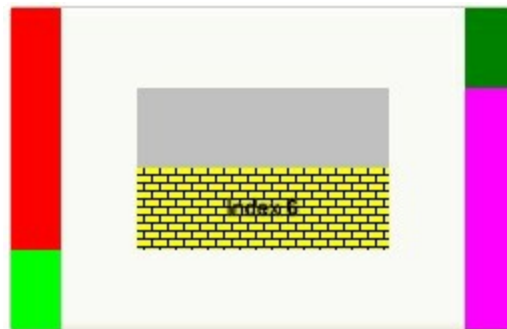
Before calling the BodyBackgroundExt property:



After calling the BodyBackgroundExt property:



and now, let's change the exPatternExt property of the object with the Index 6 to 11 ( Yard ), so finally we got:



The IndexExtEnum type supports the following values:

Name	Value	Description
exIndexExtRoot	0	Specifies the part of the object with the index 0 (root).
exIndexExt1	1	Specifies the part of the object with the index 1.
exIndexExt2	2	Specifies the part of the object with the index 2.
exIndexExt3	3	Specifies the part of the object with the index 3.
exIndexExt4	4	Specifies the part of the object with the index 4.
exIndexExt5	5	Specifies the part of the object with the index 5.
exIndexExt6	6	Specifies the part of the object with the index 6.
exIndexExt7	7	Specifies the part of the object with the index 7.

# constants PaddingEdgeInsets







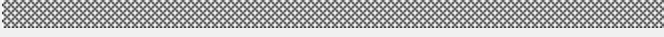





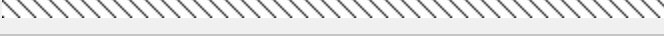



The EdgeInsets type defines the left, top, right and bottom padding to display the object. The EdgeInsets type defines the following predefined values:

Name	Value	Description
EdgeInsetsAll	-1	Indicates all margins of the object.
EdgeInsetsLeft	0	Indicates the left margin of the object.
EdgeInsetsTop	1	Indicates the top margin of the object.
EdgeInsetsRight	2	Indicates the right margin of the object.
EdgeInsetsBottom	3	Indicates the bottom margin of the object.

# constants PatternEnum

The PatternEnum type specifies the type of patterns that the element can fill with. The [Type](#) property indicates the pattern to fill the element. The [Color](#) property indicates the color to fill the element's pattern, while the [FrameColor](#) property indicates the color to show the element's border/frame if the Type property includes the exPatternFrame flag.

The PatternEnum type supports the following values:

Name	Value	Description
exPatternEmpty	0	The pattern is not visible.
exPatternSolid	1	
exPatternDot	2	
exPatternShadow	3	
exPatternNDot	4	
exPatternFDiagonal	5	
exPatternBDiagonal	6	
exPatternDiagCross	7	
exPatternVertical	8	
exPatternHorizontal	9	
exPatternCross	10	
exPatternBrick	11	
exPatternYard	12	
exPatternF2Diagonal	13	
exPatternB2Diagonal	14	
exPatternFrame	256	 . The exPatternFrame can be combined with any other value. The <a href="#">FrameColor</a> property indicates the color to show the frame.
exPatternFrameThick	768	 . The exPatternFrameThick can be combined with any other value. The <a href="#">FrameColor</a> property indicates the color to show the frame.



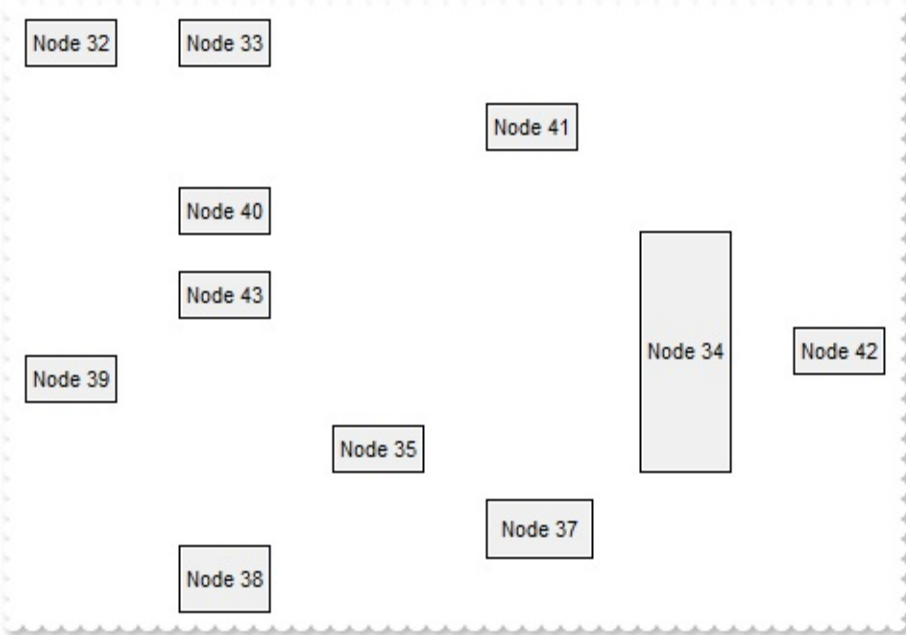
# constants **PictureDisplayEnum**

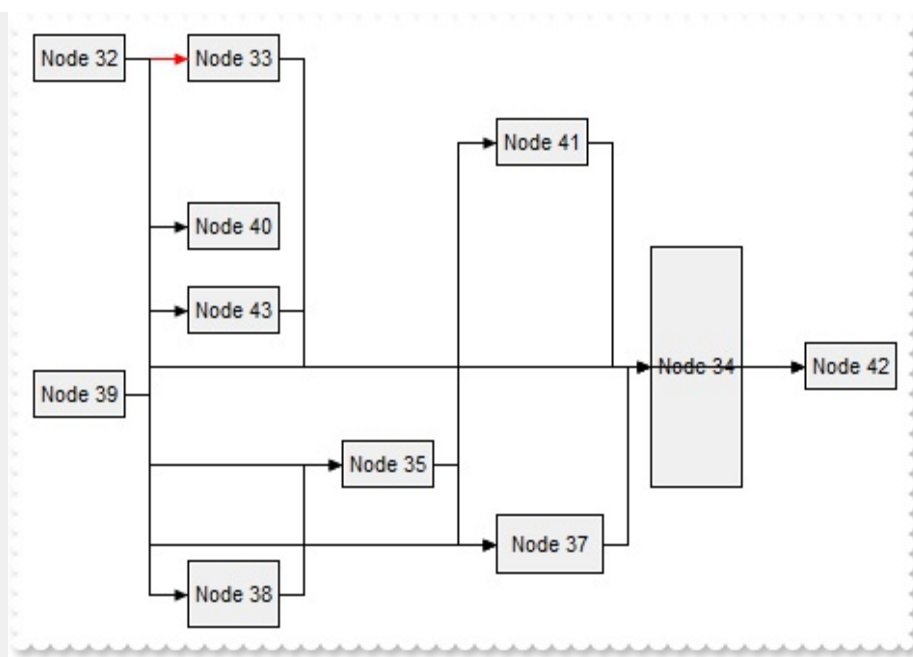
The **PictureDisplayEnum** type defines the way the control's **Picture** is arranged on the control. The [Picture](#) property assign a picture to be displayed on the control's background. The [PictureDisplay](#) property indicates how the picture is layout on the control's background. The **PictureDisplayEnum** type supports the following values:

Name	Value	Description
UpperLeft	0	The picture is vertically aligned at the top, and horizontally aligned on the left.
UpperCenter	1	The picture is vertically aligned at the top, and horizontally aligned at the center.
UpperRight	2	The picture is vertically aligned at the top, and horizontally aligned on the right.
MiddleLeft	16	The picture is vertically aligned in the middle, and horizontally aligned on the left.
MiddleCenter	17	The picture is vertically aligned in the middle, and horizontally aligned at the center.
MiddleRight	18	The picture is vertically aligned in the middle, and horizontally aligned on the right.
LowerLeft	32	The picture is vertically aligned at the bottom, and horizontally aligned on the left.
LowerCenter	33	The picture is vertically aligned at the bottom, and horizontally aligned at the center.
LowerRight	34	The picture is vertically aligned at the bottom, and horizontally aligned on the right.
Tile	48	Tiles the picture on the source.
Stretch	49	The picture is resized to fit the source.

# constants ShowExtendedLinksEnum

The ShowExtendedLinksEnum type defines how the links are shown on the surface. The [ShowLinks](#) property specifies the way the control shows the link on the surface. The ShowExtendedLinksEnum type supports the following values.

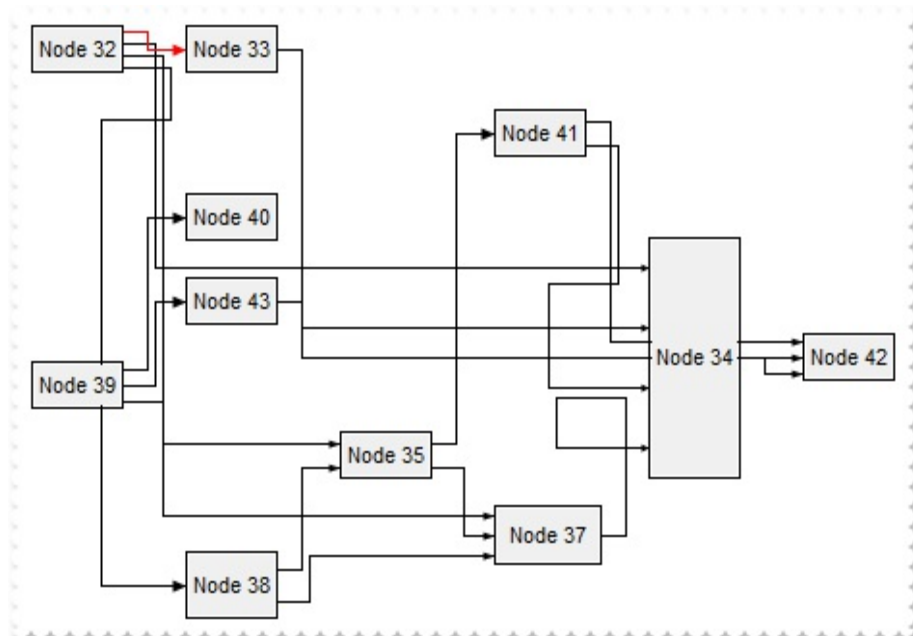
Name	Value	Description
exHideLinks	0	<div><p>Hides the links on the surface. Use the <a href="#">Visible</a> property of the Link to hide a specific link.</p></div>
exShowLinks	-1	<div><p>Shows the links on the surface. The exShowLinks flag is equivalent with exShowDefaultLinks + exShowLinksFront, it is provided for the backward compatibility only ( boolean True has the value of -1 )</p></div>



Shows the extended links on the surface. This flag is valid for rectangular links only ( `exLinkRectangular` ).

`exShowExtendedLinks`

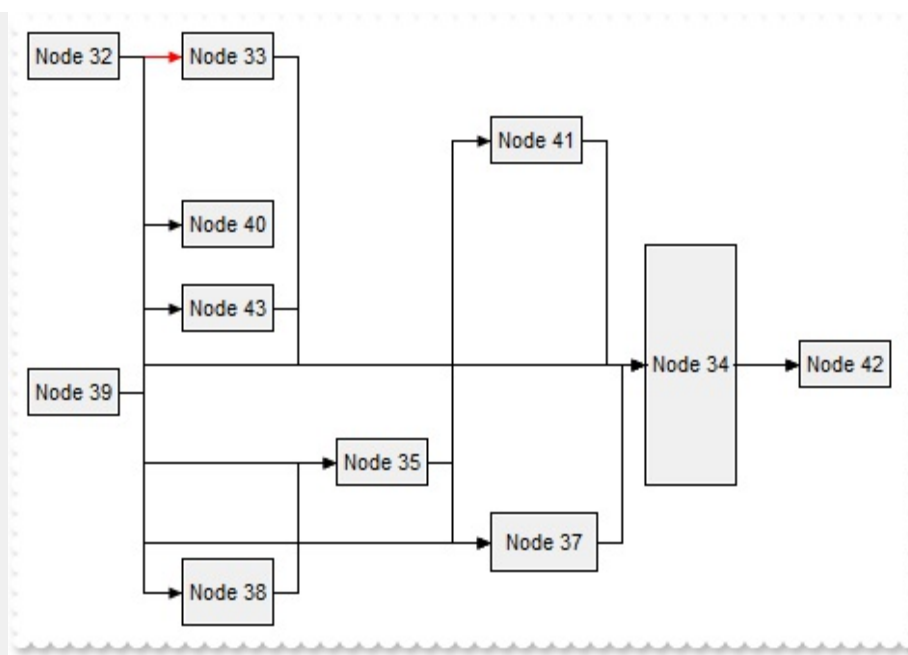
1



Shows the default links on the surface.

`exShowDefaultLinks`

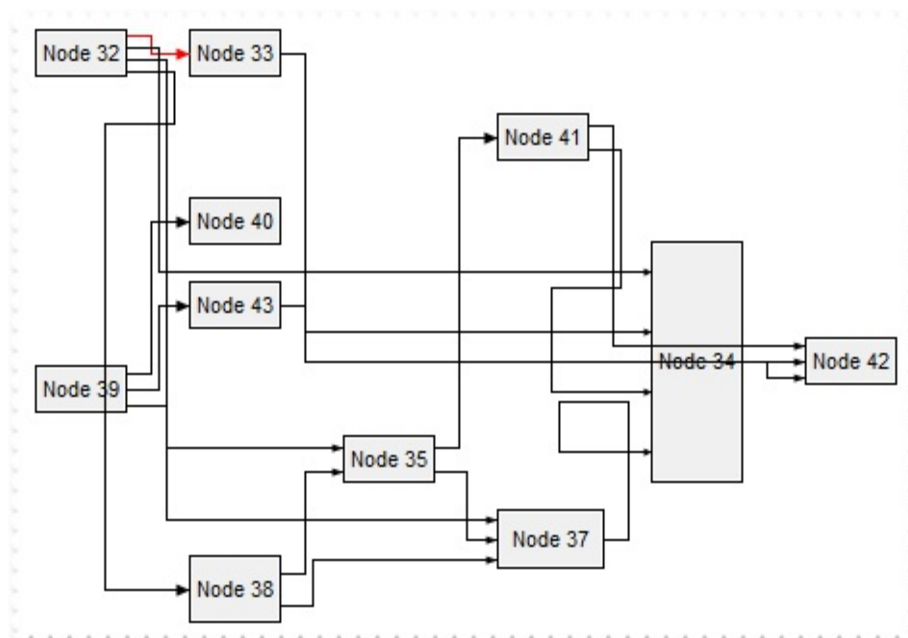
2



Shows the links on the front. This flag can be combined with the `exShowLinks`, `exShowExtendedLinks`, `exShowCrossLinksRect`, `exShowCrossLinksTriangle`, `exShowCrossLinksMixt` or `exShowDefaultLinks`. If the `exShowLinksFront` is missing, the links are shown on the control's background, so elements may show over the links.

`exShowLinksFront`

16



`exShowCrossLinksRect`

32

`exShowCrossLinksRect`. Shows rectangular cross links.

`exShowCrossLinksTriangle`

64

`exShowCrossLinksTriangle`. Shows triangular cross links.

`exShowCrossLinksMixt`

96

`exShowCrossLinksMixt`. Shows mixed cross links.



# constants ShowHandCursorOnEnum

The ShowHandCursorOnEnum type specifies the parts of the element where the hand cursor may be shown. The [ShowHandCursorOn](#) property specifies the parts of the control where the hand cursor is shown when the mouse-pointer hovers it. The [HandCursorClick](#) event occurs when the user clicks a part of the element. The ShowHandCursorOnEnum type supports the following values:

Name	Value	Description
exShowHandCursorNone	0	Specifies that no hand cursor is shown when hovering any anchor/icon/picture in the element.
exShowHandCursorPicture	1	Specifies that hand cursor is shown when hovering a picture in the element.
exShowHandCursorIcon	2	Specifies that hand cursor is shown when hovering an icon in the element.
exShowHandCursorAnchor	4	Specifies that hand cursor is shown when hovering any anchor on the element.
exShowHandCursorCheck	8	Specifies that hand cursor is shown when hovering the check-box of the element.
exShowHandCursorPictures	256	Specifies that hand cursor is shown when hovering the pictures.
exShowHandCursorExtraPictures	512	Specifies that hand cursor is shown when hovering the extra pictures.
exShowHandCursorCaption	1024	Specifies that hand cursor is shown when hovering pictures of the element's caption.
exShowHandCursorExtraCaption	2048	Specifies that hand cursor is shown when hovering pictures of the element's extra caption.
exShowHandCursorAnchorAll	3076	Specifies that the hand cursor is shown when hovering any anchor on the element.
exShowHandCursorAll	3855	Specifies that the hand cursor is shown when hovering any icon, picture, anchor, check parts of the element.

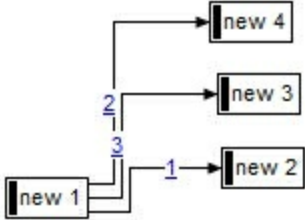
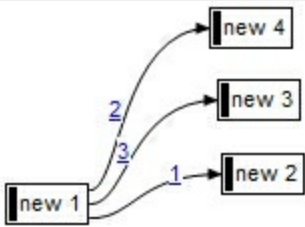
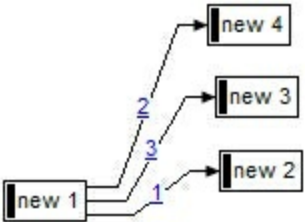
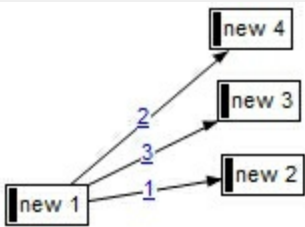
# constants ShowLinksEnum

The ShowLinksEnum type specifies the way the incoming, outgoing or collapsed links are shown. The [ShowLinksColor](#), [ShowLinksStyle](#) and [ShowLinksWidth](#) properties specifies the color, style and width of incoming, outgoing or collapsed links ( relative to selected elements ). The ShowLinksEnum type supports the following values.

Name	Value	Description
exShowLinksStartFrom	1	Shows the links that starts from selected elements. Use this flag to show the outgoing links of selected elements with a different color, style or width.
exShowLinksEndTo	2	Shows the links that ends on the selected elements. Use this flag to show the incoming links of selected elements with a different color, style or width.
exShowUnselectedLinks	4	Shows the links that are not related to any selected element.
exShowCollapsedLinks	8	Shows the collapsed links. Use this flag to show the collapsed links with a different color, style or width. The <a href="#">ShowLinksOnCollapse</a> property shows when the collapsed links ate shown or hidden.
exUpdateColorLinksOnly	16	(reserved) Prevents applying the link's color to related elements.

# constants ShowLinkTypeEnum

The ShowLinkTypeEnum type defines the type of links the control can show between element on the surface. The control's [ShowLinksType](#) property specifies the type of the links to be shown on the surface. The ShowLinkTypeEnum type supports the following values:

Name	Value	Description
exLinkRectangular	0	
exLinkRound	-1	
exLinkDirect	1	
exLinkStraight	2	



# constants UVisualThemeEnum

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

Name	Value	Description
exNoVisualTheme	0	exNoVisualTheme
exDefaultVisualTheme	16777215	exDefaultVisualTheme
exHeaderVisualTheme	1	exHeaderVisualTheme
exFilterBarVisualTheme	2	exFilterBarVisualTheme
exButtonsVisualTheme	4	exButtonsVisualTheme
exCalendarVisualTheme	8	exCalendarVisualTheme
exSliderVisualTheme	16	exSliderVisualTheme
exSpinVisualTheme	32	exSpinVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exProgressVisualTheme	128	exProgressVisualTheme
exCalculatorVisualTheme	256	exCalculatorVisualTheme

# Appearance object

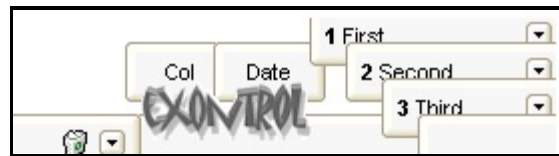
The component lets the user changes its visual appearance using **skins**, each one providing an additional visual experience that enhances viewing pleasure. Skins are relatively easy to build and put on any part of the control. The Appearance object holds a collection of skins. Use the [VisualAppearance](#) property to access the Appearance object of the control. Also, you can use the [VisualDesign](#) property to design the visual appearance of the control at design mode. 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>A string expression that indicates:</p> <ul style="list-style-type: none"><li>• an Windows XP Theme part, it should start with "XP:". For instance the <b>"XP:Header 1 2"</b> indicates the part 1 of the Header class in the state 2, in the current Windows XP theme. In this case the format of the Skin parameter should be: "XP: Control/ClassName Part State" where the ClassName defines the window/control class name in the Windows XP Theme, the Part indicates a long expression that defines the part, and the State indicates the state like listed at the end of the document. This option is available only on Windows XP that supports Themes API.</li><li>• copy of another skin with different coordinates, if it begins with "CP:". For instance, you may need to display a specified skin on a smaller rectangle. In this case, the string starts with "CP:", and contains the following <u>"CP:n l t r b"</u>, where the n is the identifier being copied, the l, t, r, and b indicate the left, top, right and bottom coordinates being used to adjust the rectangle where the skin is displayed. For instance, the <b>"CP:1 4 0 -4 0"</b>, indicates that the skin is displayed on a smaller rectangle like follows. Let's say that the control requests painting the {10, 10, 30, 20} area, a rectangle with the width of 20 pixels, and the height of 10 pixels, the skin will be displayed on the {14,10,26,20} as each coordinates in the "CP" syntax is added to the displayed rectangle, so the skin looks smaller. This way you can apply different effects to your objects in your control. The following screen shot shows the control's header when using a "CP:1 -6 -6 6 6", that displays the original skin on larger rectangles .</li></ul>
Skin as Variant	



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

A byte[] or safe arrays of VT\_I1 or VT\_UI1 expression that indicates the content of the EBN file. You can use this option when using the EBN file directly in the resources of the project. For instance, the VB6 provides the LoadResData to get the safe array o bytes for specified resource, while in VB/NET or C# the internal class Resources provides definitions for all files being inserted. ( ResourceManager.GetObject("ebn", resourceCulture) ).

## Return

## Description

Boolean

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

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



Control/ClassName		Part	States
BUTTON	BP_CHECKBOX = 3		CBS_UNCHECKED = 1 CBS_UNCHECKED = 3 CBS_UNCHECKED = 4 CBS_CHECKED = 5 CBS_CHECKEDPR CBS_CHECKEDDIS CBS_MIXEDNORM CBS_MIXEDHOT = CBS_MIXEDPRES CBS_MIXEDDISAB
		BP_GROUPBOX = 4	GBS_NORMAL = 1 GBS_DISABLED =
		BP_PUSHBUTTON = 1	PBS_NORMAL = 1 PBS_PRESSED = 2 PBS_DISABLED = PBS_DEFAULTED :
			RBS_UNCHECKED = 1 RBS_UNCHECKED = 3
		BP_RADIOBUTTON = 2	RBS_UNCHECKED = 4 RBS_CHECKED = 5 RBS_CHECKEDPR RBS_CHECKEDDIS
		BP_USERBUTTON = 5	
		CLP_TIME = 1	CLS_NORMAL = 1 CBXS_NORMAL =
			CBXS_HOT = 2 CBXS_PRESSED = CBXS_DISABLED :
			ETS_NORMAL = 1 ETS_SELECTED = 2 ETS_DISABLED = ETS_FOCUSED = ETS_READONLY =
COMBOBOX	CP_DROPDOWNBUTTON = 1		
EDIT	EP_CARET = 2		
	EP_EDITTEXT = 1		

## EXPLORERBAR

EBP\_HEADERBACKGROUND = 1

EBP\_HEADERCLOSE = 2

EBP\_HEADERPIN = 3

EBP\_IEBARMENU = 4

EBP\_NORMALGROUPBACKGROUND = 5

EBP\_NORMALGROUPCOLLAPSE = 6

EBP\_NORMALGROUPEXPAND = 7

EBP\_NORMALGROUPHEAD = 8

EBP\_SPECIALGROUPBACKGROUND = 9

EBP\_SPECIALGROUPCOLLAPSE = 10

EBP\_SPECIALGROUPEXPAND = 11

EBP\_SPECIALGROUPHEAD = 12

## HEADER

HP\_HEADERITEM = 1

HP\_HEADERITEMLEFT = 2

HP\_HEADERITEMRIGHT = 3

HP\_HEADERSORTARROW = 4

## LISTVIEW

LVP\_EMPTYTEXT = 5

LVP\_LISTDETAIL = 3

ETS\_ASSIST = 7

EBHC\_NORMAL =

EBHC\_HOT = 2

EBHC\_PRESSED =

EBHP\_NORMAL =

EBHP\_HOT = 2

EBHP\_PRESSED =

EBHP\_SELECTEDM

4 EBHP\_SELECTED

EBHP\_SELECTEDF

6

EBM\_NORMAL = 1

= 2 EBM\_PRESSEI

EBNGC\_NORMAL :

EBNGC\_HOT = 2

EBNGC\_PRESSED

EBNGE\_NORMAL :

EBNGE\_HOT = 2

EBNGE\_PRESSED

EBSGC\_NORMAL :

EBSGC\_HOT = 2

EBSGC\_PRESSED

EBSGE\_NORMAL :

EBSGE\_HOT = 2

EBSGE\_PRESSED

HIS\_NORMAL = 1

2 HIS\_PRESSED =

HILS\_NORMAL = 1

= 2 HILS\_PRESSEI

HIRS\_NORMAL = 1

= 2 HIRS\_PRESSE

HSAS\_SORTEDUP

HSAS\_SORTEDDC

LVP\_LISTGROUP = 2

LVP\_LISTITEM = 1

LVP\_LISTSORTEDDETAIL = 4

## MENU

MP\_MENUBARDROPDOWN = 4

MP\_MENUBARITEM = 3

MP\_CHEVRON = 5

MP\_MENUDROPDOWN = 2

MP\_MENUITEM = 1

MP\_SEPARATOR = 6

## MENUBAND

MDP\_NEWAPPBUTTON = 1

MDP\_SEPERATOR = 2

## PAGE

PGRP\_DOWN = 2

PGRP\_DOWNHORZ = 4

PGRP\_UP = 1

LIS\_NORMAL = 1  
2 LIS\_SELECTED =  
LIS\_DISABLED = 4  
LIS\_SELECTEDNO  
5

MS\_NORMAL = 1  
MS\_SELECTED = 2  
MS\_DEMOTED = 3  
MS\_NORMAL = 1  
MS\_SELECTED = 2  
MS\_DEMOTED = 3  
MS\_NORMAL = 1  
MS\_SELECTED = 2  
MS\_DEMOTED = 3  
MS\_NORMAL = 1  
MS\_SELECTED = 2  
MS\_DEMOTED = 3  
MS\_NORMAL = 1  
MS\_SELECTED = 2  
MS\_DEMOTED = 3  
MDS\_NORMAL = 1  
= 2 MDS\_PRESSE  
MDS\_DISABLED =  
MDS\_CHECKED =  
MDS\_HOTCHECKE

DNS\_NORMAL = 1  
= 2 DNS\_PRESSE  
DNS\_DISABLED =  
DNHZZ\_NORMAL =  
DNHZZ\_HOT = 2  
DNHZZ\_PRESSED  
DNHZZ\_DISABLED  
UPS\_NORMAL = 1  
= 2 UPS\_PRESSE  
UPS\_DISABLED =



PGRP\_UPHORZ = 3

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

**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

**SCROLLBAR**

SBP\_ARROWBTN = 1  
  
  
  
  
  
  
  
SBP\_GRIPPERHORZ = 8  
SBP\_GRIPPERVERT = 9  
  
  
SBP\_LOWERTRACKHORZ = 4  
  
  
  
SBP\_LOWERTRACKVERT = 6

ABS\_DOWNDISAB  
ABS\_DOWNHOT,  
ABS\_DOWNNORM  
ABS\_DOWNPRES  
ABS\_UPDISABLED  
ABS\_UPHOT,  
ABS\_UPNORMAL,  
ABS\_UPPRESSED,  
ABS\_LEFTDISAB  
ABS\_LEFTHOT,  
ABS\_LEFTNORMA  
ABS\_LEFTPRESSE  
ABS\_RIGHTDISAB  
ABS\_RIGHTHOT,  
ABS\_RIGHTNORM  
ABS\_RIGHTPRES

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

SBP\_THUMBBTNHORZ = 2

SBP\_THUMBBTNVERT = 3

SBP\_UPPERTRACKHORZ = 5

SBP\_UPPERTRACKVERT = 7

SBP\_SIZEBOX = 10

## SPIN

SPNP\_DOWN = 2

SPNP\_DOWNHORZ = 4

SPNP\_UP = 1

SPNP\_UPHORZ = 3

## STARTPANEL

SPP\_LOGOFF = 8

SPP\_LOGOFFBUTTONS = 9

SPP\_MOREPROGRAMS = 2

SPP\_MOREPROGRAMSARROW = 3

SPP\_PLACESLIST = 6

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  
UPS\_DISABLED =  
UPHZZ\_NORMAL :  
UPHZZ\_HOT = 2  
UPHZZ\_PRESSED  
UPHZZ\_DISABLED  
SPLS\_NORMAL =  
SPLS\_HOT = 2  
SPLS\_PRESSED =  
SPS\_NORMAL = 1  
= 2 SPS\_PRESSED

## STATUS

SPP\_PLACESLISTSEPARATOR = 7  
SPP\_PREVIEW = 11  
SPP\_PROGLIST = 4  
SPP\_PROGLISTSEPARATOR = 5  
SPP\_USERPANE = 1  
SPP\_USERPICTURE = 10  
SP\_GRIPPER = 3  
SP\_PANE = 1  
SP\_GRIPPERPANE = 2  
TABP\_BODY = 10  
TABP\_PANE = 9

## TAB

TABP\_TABITEM = 1

TABP\_TABITEMBOTHEDGE = 4

TABP\_TABITEMLEFTEDGE = 2

TABP\_TABITEMRIGHTEDGE = 3

TABP\_TOPTABITEM = 5

TABP\_TOPTABITEMBOTHEDGE = 8

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 =  
TIRES\_HOT = 2  
TIRES\_SELECTED  
TIRES\_DISABLED  
TIRES\_FOCUSED :  
TTIS\_NORMAL = 1  
= 2 TTIS\_SELECTE  
TTIS\_DISABLED =  
TTIS\_FOCUSED =  
TTIBES\_NORMAL :  
TTIBES\_HOT = 2  
TTIBES\_SELECTE  
TTIBES\_DISABLED  
TTIBES\_FOCUSED  
TTILES\_NORMAL :

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

TP\_DROPDOWNBUTTON = 2

TP\_SPLITBUTTON = 3

TP\_SPLITBUTTONDROPDOWN = 4

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  
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

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

TKP\_THUMBBOTTOM = 4

TKP\_THUMBLEFT = 7

TKP\_THUMBRIGHT = 8

TS\_NORMAL = 1  
TS\_PRESSED = 3  
TS\_DISABLED = 4  
TS\_CHECKED = 5  
TS\_HOTCHECKED  
TS\_NORMAL = 1  
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  
TUBS\_PRESSED =  
TUBS\_FOCUSED =  
TUBS\_DISABLED =  
TUVLS\_NORMAL =  
TUVLS\_HOT = 2  
TUVLS\_PRESSED  
TUVLS\_FOCUSED  
TUVLS\_DISABLED  
TUVRS\_NORMAL =  
TUVRS\_HOT = 2  
TUVRS\_PRESSED  
TUVRS\_FOCUSED  
TUVRS\_DISABLED  
TUTS\_NORMAL =

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

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

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

CBS\_NORMAL = 1

= 2 CBS\_PUSHED

CBS\_DISABLED =

FS\_ACTIVE = 1 FS

= 2

FS\_ACTIVE = 1 FS

= 2

FS\_ACTIVE = 1 FS

= 2

WP\_HELPBUTTON = 23

WP\_HORZSCROLL = 25

WP\_HORZTHUMB = 26

WP\_MAX\_BUTTON

WP\_MAXCAPTION = 5

WP\_MDICLOSEBUTTON = 20

WP\_MDIHELPBUTTON = 24

WP\_MDIMINBUTTON = 16

WP\_MDIRESTOREBUTTON = 22

WP\_MDISYSBUTTON = 14

WP\_MINBUTTON = 15

WP\_MINCAPTION = 3

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

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

WP_RESTOREBUTTON = 21	= 2 RBS_PUSHED RBS_DISABLED =
WP_SMALLCAPTION = 2	CS_ACTIVE = 1 CS = 2 CS_DISABLED
WP_SMALLCAPTIONSIZINGTEMPLATE = 31	
WP_SMALLCLOSEBUTTON = 19	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
WP_SMALLFRAMEBOTTOM = 12	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMEBOTTOMSIZINGTEMPLATE = 37	
WP_SMALLFRAMELEFT = 10	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMELEFTSIZINGTEMPLATE = 33	
WP_SMALLFRAMERIGHT = 11	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMERIGHTSIZINGTEMPLATE = 35	
WP_SMALLHELPBUTTON	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_SMALLMAXBUTTON	MAXBS_NORMAL : MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_SMALLMAXCAPTION = 6	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
WP_SMALLMINCAPTION = 4	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_SMALLRESTOREBUTTON	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_SMALLSYSBUTTON	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED = SBS_NORMAL = 1



WP\_SYSBUTTON = 13

= 2 SBS\_PUSHED

SBS\_DISABLED =

VSS\_NORMAL = 1

WP\_VERTSCROLL = 27

= 2 VSS\_PUSHED

VSS\_DISABLED =

VTs\_NORMAL = 1

WP\_VERTTHUMB = 28

2 VTs\_PUSHED =

VTs\_DISABLED =

# 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.

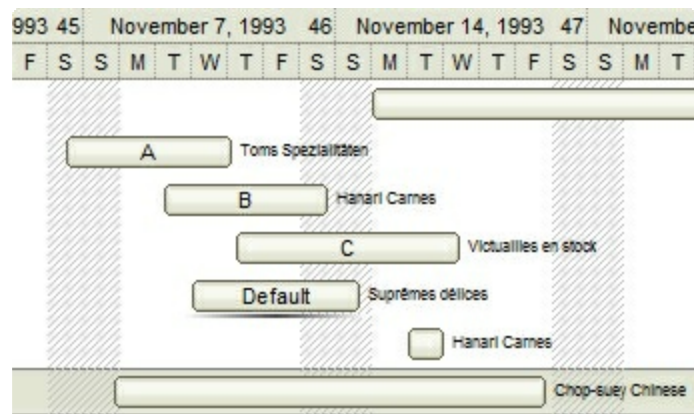
Click here  to watch a movie on how you can change the colors to be applied on EBN objects.

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

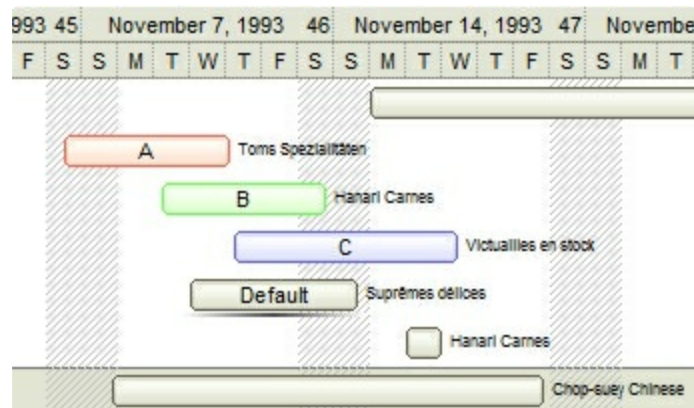
- "A" in Red ( RGB(255,0,0 ) , for instance the bar's property exBarColor is 0x10000FF
- "B" in Green ( RGB(0,255,0 ) , for instance the bar's property exBarColor is 0x100FF00
- "C" in Blue ( RGB(0,0,255 ) , for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for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

The RenderType property could be one of the following:

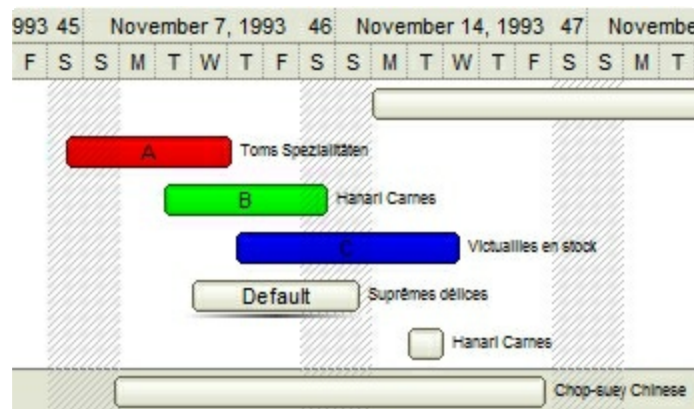
- **-3**, *no color is applied*. For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color ( Blue color ) is ignored. You can use this option to allow the control displays the EBN colors or not.



- **-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), ... )

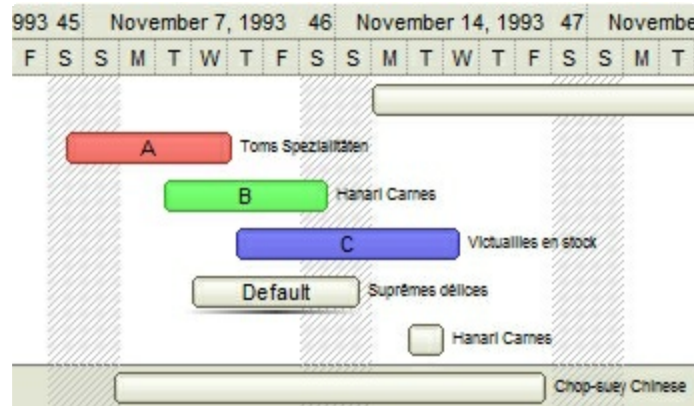


- **-1, AND-color scheme,** The color to be applied on the part of the control is an AND bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the AND bit for the entire Blue channel, or in other words, it applies a more Blue to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



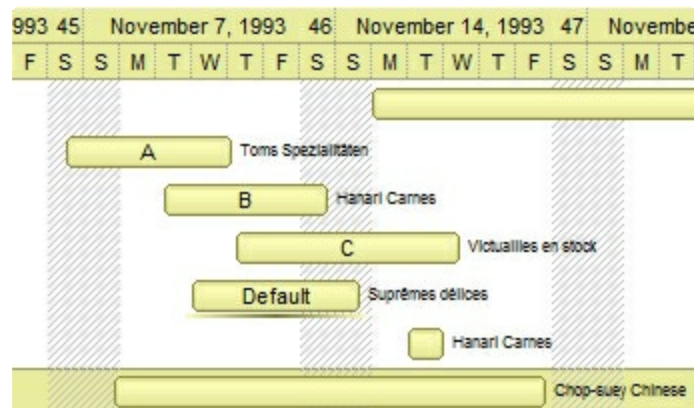
- **0, default,** the specified color is applied to the EBN. For instance, the

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.

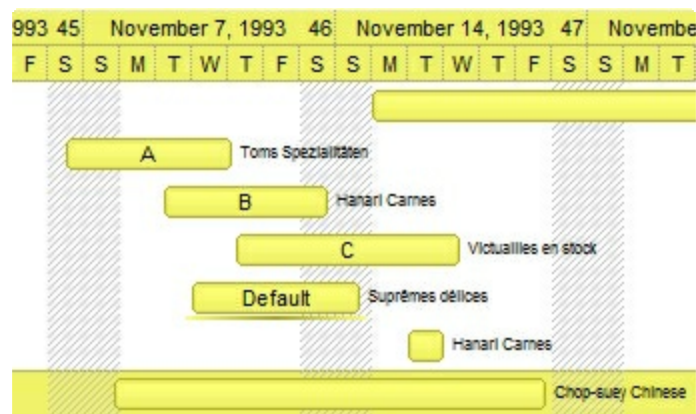


- **0xAABBGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on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

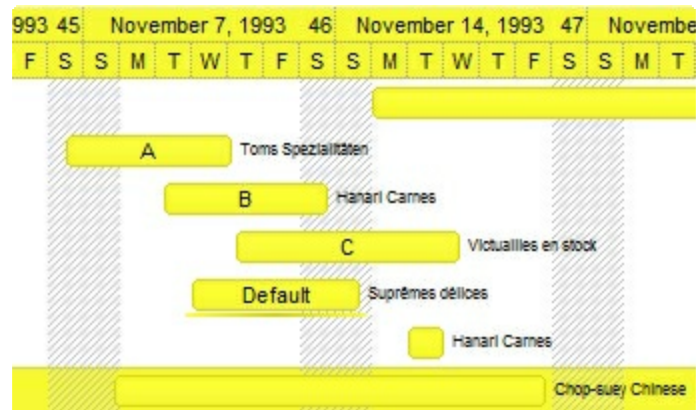
*The following picture shows the control with the RenderType property on 0x4000FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256 ):*



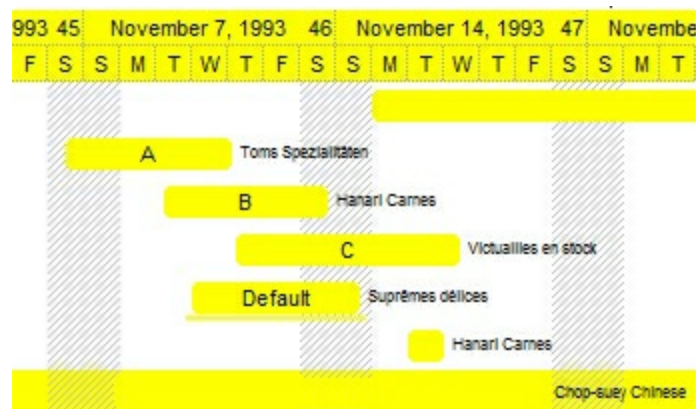
*The following picture shows the control with the RenderType property on 0x8000FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256 ):*



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 ):



# Element object

The Element object defines an object on the surface. The Element object supports the following properties and methods:

Name	Description
<a href="#">AllowChangeParent</a>	Gets or sets a value that indicates whether the element can change its parent ( by drag and drop ).
<a href="#">AllowInsertChild</a>	Gets or sets a value that indicates whether the element allows child elements ( by drag and drop ).
<a href="#">AutoHeight</a>	Gets the height of the element to fit its content ( as if the AutoSize property is True ).
<a href="#">AutoSize</a>	Specifies if the element computes its size automatically.
<a href="#">AutoWidth</a>	Gets the width of the element to fit its content ( as if the AutoSize property is True ).
<a href="#">BackColor</a>	Gets or sets a value that indicates the element's background color.
<a href="#">BackgroundExt</a>	Indicates additional colors, text, images that can be displayed on the element's background using the EBN string format.
<a href="#">BackgroundExtValue</a>	Specifies at runtime, the value of the giving property for specified part of the background extension.
<a href="#">BorderColor</a>	Gets or sets a value that indicates the element's border color.
<a href="#">BorderPadding</a>	Returns or sets a value that indicates the padding of the element's borders.
<a href="#">BringToFront</a>	Brings the element to front.
<a href="#">Caption</a>	Gets or sets a value that indicates the HTML caption to be displayed on the element.
<a href="#">CaptionAlign</a>	Indicates the alignment of the element's caption.
<a href="#">CaptionSingleLine</a>	Specifies if the element's caption is displayed on single or multiple lines.
<a href="#">CheckBoxAlign</a>	Indicates the alignment of the element's checkbox.
<a href="#">Checked</a>	Gets or sets a value that indicates the element's check-box state.
<a href="#">ChildCount</a>	Counts the number of child elements.



<a href="#">ChildPosition</a>	Specifies the position of the element while it is a child element.
<a href="#">Children</a>	Returns a safe array of child elements.
<a href="#">ClientPadding</a>	Returns or sets a value that indicates the padding of the element's client.
<a href="#">Control</a>	Specifies the identifier of the inner control hosted by the current element.
<a href="#">Edit</a>	Edits the element.
<a href="#">ElementFormat</a>	Specifies the way the control shows the parts of the element.
<a href="#">Enabled</a>	Gets or sets a value that indicates whether the element is enabled or disabled.
<a href="#">EndUpdateElement</a>	Adds programmatically updated properties of the element to undo/redo queue.
<a href="#">EnsureVisible</a>	Scrolls the surface to ensure that the current element fits the control's visible area.
<a href="#">Expanded</a>	Expands or collapses the element.
<a href="#">ExtraCaption</a>	Gets or sets a value that indicates the extra HTML caption to be displayed on the element.
<a href="#">ExtraCaptionAlign</a>	Indicates the alignment of the element's extra caption.
<a href="#">ExtraCaptionSingleLine</a>	Specifies if the element's extra caption is displayed on single or multiple lines.
<a href="#">ExtraPictures</a>	Specifies the list of extra pictures to be displayed on the element.
<a href="#">ExtraPicturesAlign</a>	Indicates the alignment of the element's extra picture.
<a href="#">FirstChild</a>	Gets the first child of the element.
<a href="#">ForeColor</a>	Gets or sets a value that indicates the element's foreground color.
<a href="#">Height</a>	Specifies the height of the element.
<a href="#">ID</a>	Specifies the element's unique identifier.
<a href="#">IncomingLinks</a>	Returns a safe array of incoming links.
<a href="#">InflateSize</a>	Increases or decreases the width and height of the element.
<a href="#">LastChild</a>	Gets the last child of the element.
<a href="#">Level</a>	Specifies the level of the element in a hierarchy.

[License](#)

Indicates the runtime license required to create the inner control.

[MaxHeight](#)

Specifies the maximum height of the element.

[MaxWidth](#)

Specifies the maximum width of the element.

[MinHeight](#)

Specifies the minimum height of the element.

[MinWidth](#)

Specifies the minimum width of the element.

[MoveTo](#)

Moves the element to a new position.

[NextSiblingChild](#)

Retrieves the next sibling of the element in the parent's child list

[NextVisibleChild](#)

Retrieves the next visible element in the parent's child list

[Object](#)

Returns the inner object hosted by the current element.

[OutgoingLinks](#)

Returns a safe array of outgoing links.

[OverviewColor](#)

Gets or sets a value that indicates the element's overview color.

[Padding](#)

Returns or sets a value that indicates the padding of the element's background.

[Parent](#)

Specifies the element's parent.

[PathTo](#)

Determines if there is any path from the current element to the specified element.

[Pattern](#)

Specifies the pattern to be shown on the element's background.

[Picture](#)

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

[PictureDisplay](#)

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

[Pictures](#)

Specifies the list of pictures to be displayed on the element.

[PicturesAlign](#)

Indicates the alignment of the element's picture.

[PrevSiblingChild](#)

Retrieves the prev sibling of the element in the parent's child list

[PrevVisibleChild](#)

Retrieves the prev visible element in the parent's child list

[Resizable](#)

Gets or sets a value that indicates whether the user can resize the element.

[ScrollTo](#)

Moves or scrolls the surface, so the current element aligns to the specified corner.

<a href="#">Selectable</a>	Indicates if the element is selectable.
<a href="#">Selected</a>	Indicates if the element is selected or unselected.
<a href="#">SendToBack</a>	Sends the element to the back.
<a href="#">ShowCheckBox</a>	Gets or sets a value that indicates whether the element shows or hides the check-box.
<a href="#">ShowHandCursorOn</a>	Specifies whether the hand cursor is shown when hovering the element.
<a href="#">StartUpdateElement</a>	Starts changing properties of the element, so EndUpdateElement method adds programmatically updated properties to undo/redo queue.
<a href="#">StatusAlign</a>	Specifies the alignment of the status inside the element.
<a href="#">StatusColor</a>	Gets or sets a value that indicates the element's status color.
<a href="#">StatusPadding</a>	Returns or sets a value that indicates the padding of the element's status.
<a href="#">StatusPattern</a>	Specifies the pattern of the element's status
<a href="#">StatusSize</a>	Specifies the size of the status inside the element.
<a href="#">ToolTip</a>	Gets or sets a value (tooltip) that's displayed once the cursor hovers the element.
<a href="#">ToolTipTitle</a>	Gets or sets a value (title) that's displayed once the cursor hovers the element.
<a href="#">Type</a>	Specifies the element's type.
<a href="#">UserData</a>	Indicates any extra data associated with the element.
<a href="#">Visible</a>	Shows or hides the element.
<a href="#">VisibleChildCount</a>	Counts the number of visible child elements.
<a href="#">VisibleChildren</a>	Returns a safe array of visible child elements.
<a href="#">Width</a>	Specifies the width of the element.
<a href="#">Window</a>	Returns or sets the handle of the window to be hosted by the element.
<a href="#">X</a>	Specifies the element's x-position.
<a href="#">Y</a>	Specifies the element's y-position.

# property Element.AllowChangeParent as Boolean

Gets or sets a value that indicates whether the element can change its parent ( by drag and drop ).

Type	Description
Boolean	A Boolean expression that specifies whether the element can change its parent at runtime ( by drag and drop ).

By default, the AllowChangeParent property is True. Use the AllowChangeParent property to prevent changing the element's Parent at runtime. The [AllowInsertChild](#) property specifies whether other elements can be inserted as child elements of the current element. The [Parent](#) property indicates the element's parent. Use the [AllowInsertObject](#) property to specify whether the elements can be be dropped over other elements to change its parent or the children list. The [Children](#) property specifies the list of child elements.

# property Element.AllowInsertChild as Boolean

Gets or sets a value that indicates whether the element allows child elements ( by drag and drop ).

Type	Description
Boolean	A Boolean expression that indicates whether the element allows child elements ( by drag and drop )

The AllowInsertChild property specifies whether other elements can be inserted as child elements of the current element. Use the [AllowChangeParent](#) property to prevent changing the element's Parent at runtime. The [Parent](#) property indicates the element's parent. Use the [AllowInsertObject](#) property to specify whether the elements can be be dropped over other elements to change its parent or the children list. The [Children](#) property specifies the list of child elements.

# property Element.AutoHeight as Long

Gets the height of the element to fit its content ( as if the AutoSize property is True ).

Type	Description
Long	A long expression that specifies the height required to display the element's content.

The AutoHeight property gets the height of the element's content as if the [AutoSize](#) property is True. The [AutoWidth](#) property gets the width required to display the element's content. The [Width](#) / [Height](#) property specifies the width / height of the element while the [AutoSize](#) property is False.

# property Element.AutoSize as Boolean

Specifies if the element computes its size automatically.

Type	Description
Boolean	A Boolean expression that specifies whether the size of the element based on its content.

By default, the AutoSize property is True, if it the [Type](#) is exElementHostDefault. While the AutoSize property is True, the element is not resizable. Its size is changed based on the element's content: caption, extra-caption, icons, and so on. The [InflateSize](#) property indicates the size to be added to the default auto-size for increasing the size of the element. The [Padding](#), [BorderPadding](#) and [StatusPadding](#) properties specifies the padding to be applied on client, border and status parts of the element. The [Caption](#) property specifies the element's caption. The [ExtraCaption](#) property specifies the element's extra caption. The [ShowCheckBox](#) property indicates whether the element's checkbox is visible or hidden. If AutoSize property is False, you can use the [Resizable](#) on False to prevent resizing an element. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element. Use the [AllowResizeObject](#) property to specify whether the element can be resized at runtime.

# property Element.AutoWidth as Long

Gets the width of the element to fit its content ( as if the AutoSize property is True ).

Type	Description
Long	A long expression that specifies the width required to display the element's content.

The AutoWidth property gets the height of the element's content as if the [AutoSize](#) property is True. The [AutoHeight](#) property gets the height required to display the element's content. The [Width](#) / [Height](#) property specifies the width / height of the element while the [AutoSize](#) property is False.



# property Element.BackColor as Color

Gets or sets a value that indicates the element's background color.

Type	Description
Color	A Color expression that specifies the color to show element's background. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used to paint the part. Use the <a href="#">Add</a> method to add new skins to the control. The -1 indicates that the element's background is transparent.

By default, the BackColor property is -1, which indicates that the element's background is transparent. The BackColor property specifies the element's background color. [Background](#)(exElementBackColor) property specifies the default background color / visual appearance. The [ForeColor](#) property specifies the element's foreground color. The [Pattern](#) property defines the pattern to be shown on the control's background. The [Padding](#) property defines the padding of the element. The [BorderColor](#) property specifies the color to show the border for a specific element. The [StatusColor](#) property specifies the color or the visual appearance to show the element's status part. The [Picture](#) property specifies the picture to be shown on the element's background. The [PictureDisplay](#) property specifies the way the element's picture is displayed on the element's background.

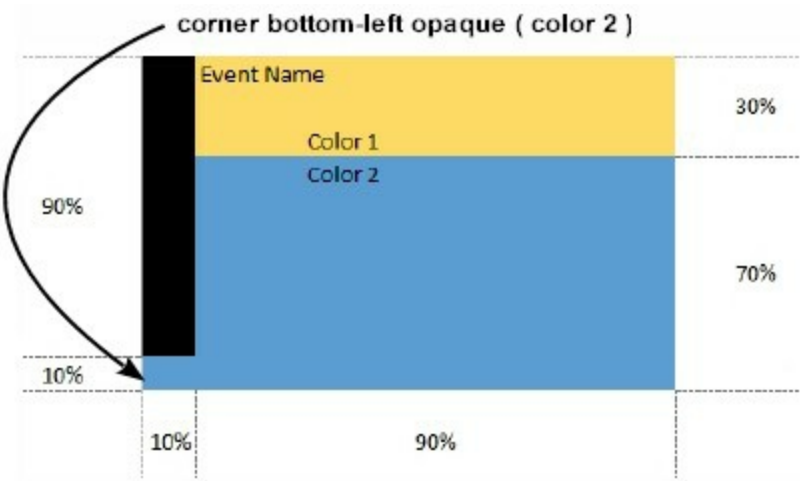
# property Element.BackgroundColor as String

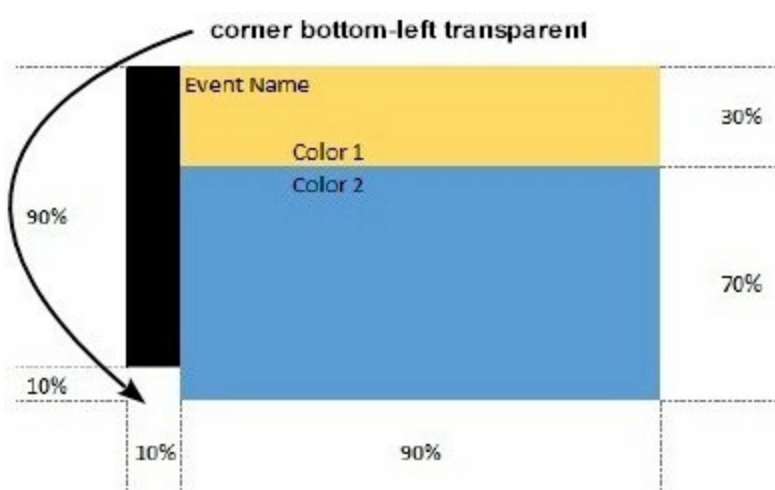
Indicates additional colors, text, images that can be displayed on the object's background using the EBN string format.

Type	Description
String	A String expression ( "EBN String Format" ) that defines the layout of the UI to be applied on the object's background. The <a href="#">syntax</a> of EBN String Format in BNF notation is shown bellow. <i>You can use the EBN's Builder of eXButton/COM control to define visually the EBN String Format.</i>

By default, the BodyBackgroundColor property is empty. Using the BodyBackgroundColor property you have unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the object's background. *For instance, let's say you need to display **more** colors on the object's background, or just want to display an **additional** caption or image to a specified location on the object's background.* The EBN String Format defines the parts of the EBN to be applied on the object's background. The [EBN](#) is a set of UI elements that are built as a tree where each element is anchored to its parent element. Use the [BackgroundColorValue](#) property to change at runtime any UI property for any part that composes the EBN String Format. The BodyBackgroundColor property is applied right after setting the object's bgcolor, and before drawing the default object's captions, icons or pictures.

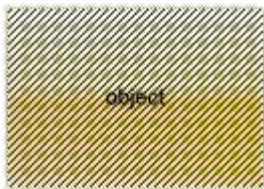
Complex samples:



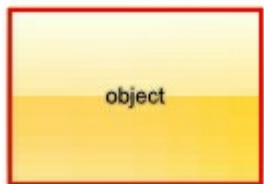


Easy samples:

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



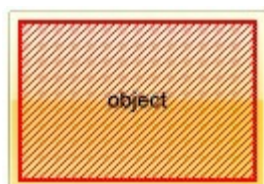
- "[frame=RGB(255,0,0),framethick]", draws a red thick-border around the object.



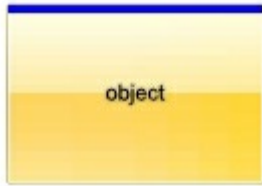
- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a red thick-border around the object, with a patten inside.



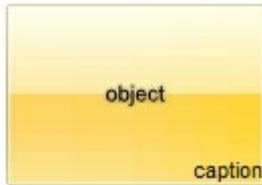
- "[[patterncolor=RGB(255,0,0)]  
(none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(255,0,0),framethick])]", draws a red thick-border around the object, with a patten inside, with a 4-pixels wide padding:



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



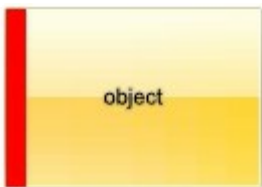
- "[text=`caption`,align=0x22]", shows the caption string aligned to the bottom-right side of the object's background.



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



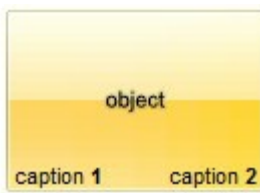
- "left[10,back=RGB(255,0,0)]", draws a red line on the left side of the object's background, of 10-pixels wide.



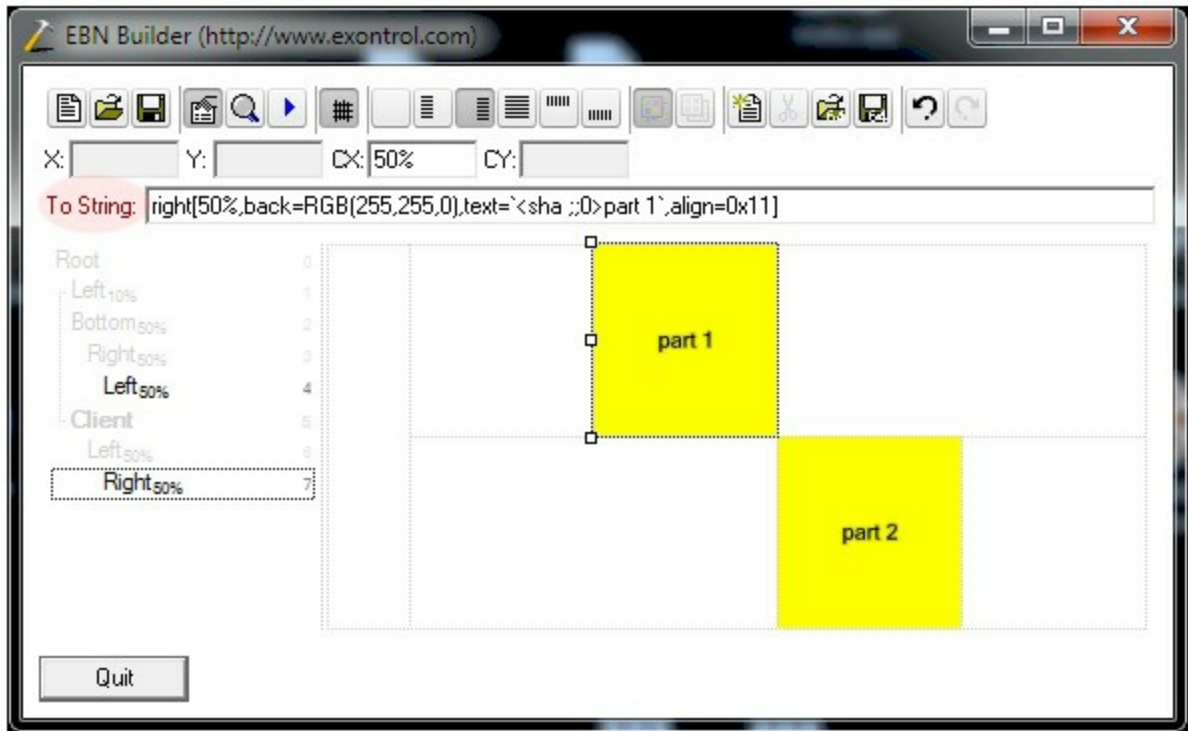
- "bottom[50%,pattern=6,frame]", shows the [BDiagonal](#) pattern with a border around on the lower-half part of the object's background.



- "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



The Exontrol's [eXButton](http://www.exontrol.com) WYSWYG Builder helps you to generate or view the EBN String Format, in the **To String** field as shown in the following screen shot:



The **To String** field of the EBN Builder defines the **EBN String Format** that can be used on BodyBackgroundExt property.

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> ::= <backcolor> | <text> | <wordwrap> | <align> | <pattern> |
<patterncolor> | <frame> | <framethick> | <data> | <others>
<equal> ::= "="
  
```

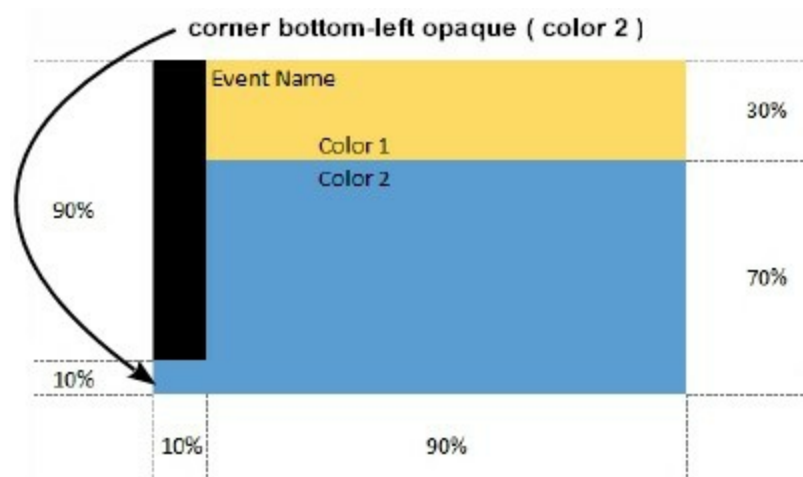
```

<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.*

Now, let's say we have the following request to layout the colors on the objects:

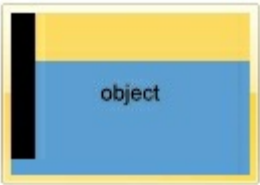


We define the BodyBackgroundExt property such as  
 "top[30%,back=RGB(253,218,101)],client[back=RGB(91,157,210)],none[(0%,0%,10%,100%  
 (top[90%,back=RGB(0,0,0)])", and it looks as:

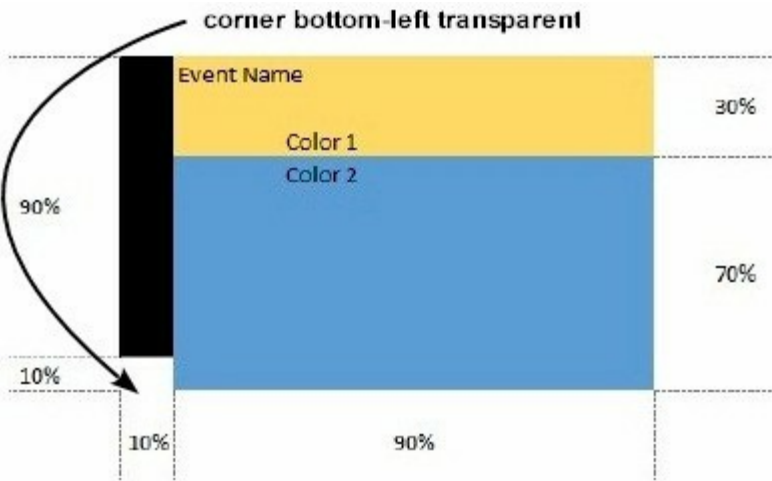
```
To String: top[30%,back=RGB(253,218,101)],client[back=RGB(91,157,210)],none([0%,0%,10%,100%])(top[90%,back=RGB(0,0,0)])
```



so, if we apply to our object we got:

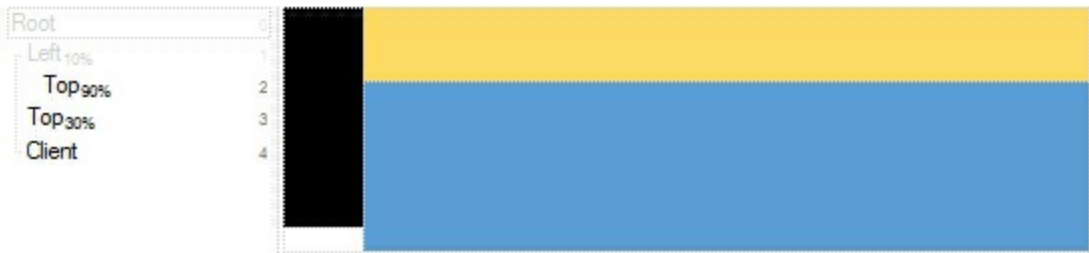


Now, lets say we have the following request to layout the colors on the objects:

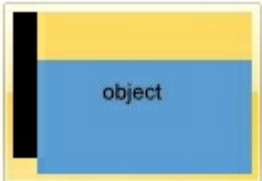


We define BodyBackgroundExt property such as "left[10%](top[90%,back=RGB(0,0,0)]),top[30%,back=RGB(254,217,102)],client[back=RGB(91,156,212)]" and it looks as:

```
To String: left[10%](top[90%,back=RGB(0,0,0)]),top[30%,back=RGB(254,217,102)],client[back=RGB(91,156,212)]
```



so, if we apply to our object we got:



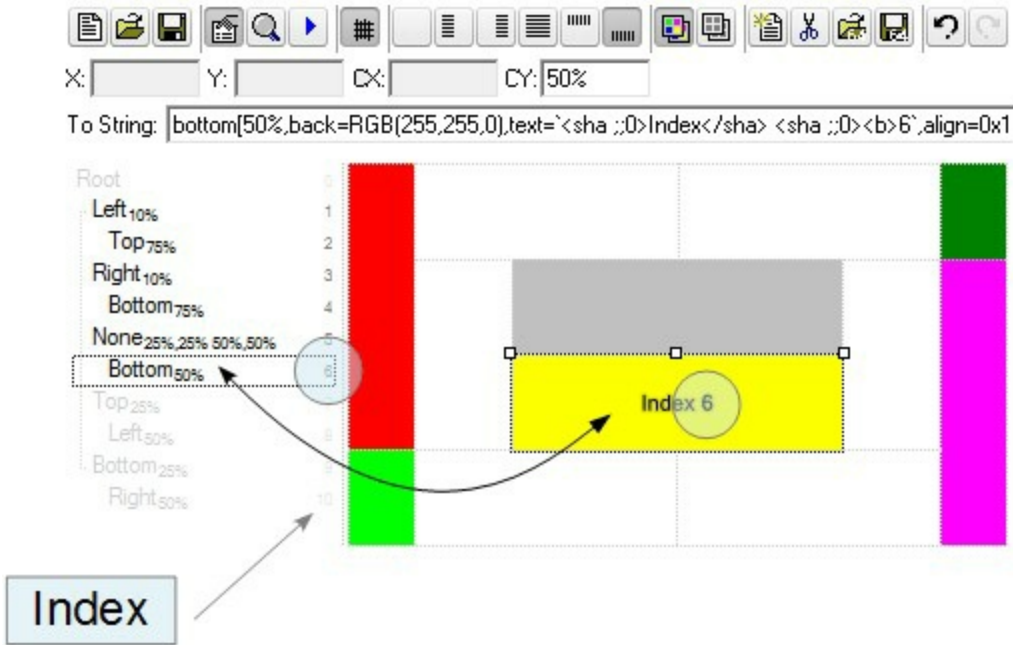


# property Element.BackgroundImageValue(Index as IndexExtEnum, Property as BackgroundExtPropertyEnum) as Variant

Specifies at runtime, the value of the giving property for specified part of the background extension.

Type	Description
	A Long expression that defines the index of the part that composes the EBN to be accessed / changed.
	The following screen shot shows where you can find Index of the parts:

Index as IndexExtEnum



The screen shot shows that the EBN contains 11 elements, so in this case the Index starts at 0 ( root element ) and ends on 10.

Property as <a href="#">BackgroundExtPropertyEnum</a>	A <a href="#">BackgroundExtPropertyEnum</a> expression that specifies the property to be changed as explained bellow.
Variant	A Variant expression that defines the part's value. The Type of the expression depending on the Property parameter as explained bellow.

Use the BackgroundExtValue property to change at runtime any UI property for any part that composes the EBN String Format. The BackgroundExtValue property has no effect if the [BodyBackgroundExt](#) property is empty ( by default ). *The idea is as follows: first you need to decide the layout of the UI to put on the object's background, using the*

*BodyBackgroundExt* property, and next ( if required ), you can change any property of any part of the background extension to a new value. In other words, let's say you have the same layout to be applied to some of your objects, so you specify the *BodyBackgroundExt* to be the same for them, and next use the *BackgroundExtValue* property to change particular properties ( like back-color, size, position, anchor ) for different objects.

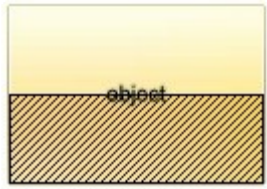
You can access/define/change the following UI properties of the element:

- **exBackColorExt**(1), Indicates the background color / EBN color to be shown on the part of the object. *Sample: 255 indicates red, RGB(0,255,0) green, or 0x1000000. (Color/Numeric expression, The last 7 bits in the high significant byte of the color indicate the identifier of the skin being used )*
- **exClientExt**(2), Specifies the position/size of the object, depending on the object's anchor. The syntax of the *exClientExt* is related to the *exAnchorExt* value. *For instance, if the object is anchored to the left side of the parent ( exAnchorExt = 1 ), the exClientExt specifies just the width of the part in pixels/percents, not including the position. In case, the exAnchorExt is client, the exClientExt has no effect. Sample: 50% indicates half of the parent, 25 indicates 25 pixels, or 50%-8 indicates 8-pixels left from the center of the parent. (String/Numeric expression)*
- **exAnchorExt**(3), Specifies the object's alignment relative to its parent. *(Numeric expression)*
- **exTextExt**(4), Specifies the HTML text to be displayed on the object. *(String expression)*
- **exTextExtWordWrap**(5), Specifies that the object is wrapping the text. The *exTextExt* value specifies the HTML text to be displayed on the part of the EBN object. This property has effect only if there is a text assigned to the part using the *exTextExt* flag. *(Boolean expression)*
- **exTextExtAlignment**(6), Indicates the alignment of the text on the object. The *exTextExt* value specifies the HTML text to be displayed on the part of the EBN object. This property has effect only if there is a text assigned to the part using the *exTextExt* flag *(Numeric expression)*
- **exPatternExt**(7), Indicates the pattern to be shown on the object. The *exPatternColorExt* specifies the color to show the pattern. *(Numeric expression)*
- **exPatternColorExt**(8), Indicates the color to show the pattern on the object. The *exPatternColorExt* property has effect only if the *exPatternExt* property is not 0 ( empty ). The *exFrameColorExt* specifies the color to show the frame ( the *exPatternExt* property includes the *exFrame* or *exFrameThick* flag ). *(Color expression)*
- **exFrameColorExt**(9), Indicates the color to show the border-frame on the object. This property set the *Frame* flag for *exPatternExt* property. *(Color expression)*
- **exFrameThickExt**(11), Specifies that a thick-frame is shown around the object. This property set the *FrameThick* flag for *exPatternExt* property. *(Boolean expression)*
- **exUserDataExt**(12), Specifies an extra-data associated with the object. *(Variant*

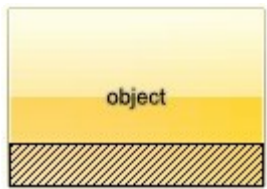
*expression)*

For instance, having the BodyBackgroundExt on "bottom[50%,pattern=6,frame]"

we got:



so let's change the percent of 50% to 25% like BackgroundExtValue(1,2) on "25%", where 1 indicates the first element after root, and 2 indicates the **exClientExt** property, we get:



In VB you should have the following syntax:

```
.BodyBackgroundExt = "bottom[50%,pattern=6,frame]"  
.BackgroundExtValue(exIndexExt1, exClientExt) = "25%"
```

# property Element.BorderColor as Color

Gets or sets a value that indicates the element's border color.

Type	Description
Color	A Color expression that specifies the color to show element's border. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used to paint the part. Use the <a href="#">Add</a> method to add new skins to the control.

By default, the BorderColor is -1, which indicates that the default border color is applied. The [Background](#)(exElementBorderColor) property specifies the default border color / visual appearance. Use the BorderColor property to specify a different border color. The [BorderPadding](#) property specifies the border padding. The [BackColor](#) property specifies the element's background color. The [ForeColor](#) property specifies the element's foreground color.

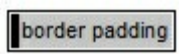
# property Element.BorderPadding(Edge as PaddingEdgeEnum) as Long

Returns or sets a value that indicates the padding of the element's borders.

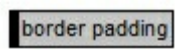
Type	Description
Edge as <a href="#">PaddingEdgeEnum</a>	A PaddingEdgeEnum expression that specifies the edge to be changed
Long	A long expression that defines the padding

By default, BorderPadding property is 1. The BorderPadding property specifies the padding to be applied on borders, to define the position of the status and the body parts of the element. The [StatusPadding](#) property specifies the status padding. The [Padding](#) property specifies the body padding.

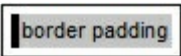
The following screen shot shows the element when BorderPadding is 1 (default):



The following screen shot shows the element when BorderPadding is 0:



The following screen shot shows the element when BorderPadding is 4:



# method Element.BringToFront ()

Brings the element to front.

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

The BringToFront method brings the element to the front. The [SendToBack](#) method sends the current element to the back. For instance, if two element gets intersected, you can use the BringToFront method to bring one element on front, or [SendToBack](#) method to send the element on the back. The BringToFront and [SendToBack](#) methods changes the drawing order of the elements.

# property Element.Caption as String

Gets or sets a value that indicates the HTML caption to be displayed on the element.

Type	Description
String	A String expression that defines the HTML caption to be displayed on the element's background.

By default, the Caption property is empty. Use the Caption property to define the label or caption to be displayed on the element's background. The [CaptionAlign](#) property specifies the alignment of the caption relative to the edges of the element. The [CaptionSingleLine](#) property specifies whether the element's caption is displayed on single or multiple lines. The [ExtraCaption](#) property defines the second or the extra caption to be displayed on the element's background. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface. Use the [Pictures](#) property to display one or more icons, picture to the element. Use the [ElementFormat](#) property to define how the parts of the element are displayed.

The Caption property supports the following HTML elements:

- **<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.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using <a ;exp=> or <a ;e64=> anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu</a>" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY

string encodes the "<fgcolor 808080>show lines<a>-</a></fgcolor>" The Decode64Text/Encode64Text methods of the exPrint can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **<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 Element.CaptionAlign as ContentAlignmentEnum

Indicates the alignment of the element's caption.

Type	Description
<a href="#">ContentAlignmentEnum</a>	<p>A ContentAlignmentEnum expression that specifies the alignment of the element's caption relative to is edges. The CaptionAlign property supports additionally the following flag:</p> <ul style="list-style-type: none"><li>• exWidth (4), to distribute the text on the element's width</li></ul>

The CaptionAlign property specifies the alignment of the caption relative to the edges of the element. The [CaptionSingleLine](#) property specifies whether the element's caption is displayed on single or multiple lines. Use the [Caption](#) property to define the label or caption to be displayed on the element's background. The [ExtraCaption](#) property defines the second or the extra caption to be displayed on the element's background. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

# property Element.CaptionSingleLine as CaptionSingleLineEnum

Specifies if the element's caption is displayed on single or multiple lines.

Type	Description
<a href="#">CaptionSingleLineEnum</a>	A CaptionSingleLineEnum expression that specifies whether the element's caption is displayed on single or multiple lines.

By default, the CaptionSingleLine property is exCaptionSingleLine. The CaptionSingleLine property specifies whether the element's caption is displayed on single or multiple lines. Use the [Caption](#) property to define the label or caption to be displayed on the element's background. The [CaptionAlign](#) property specifies the alignment of the caption relative to the edges of the element. The [ExtraCaption](#) property defines the second or the extra caption to be displayed on the element's background. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

# property Element.CheckBoxAlign as ContentAlignmentEnum

Indicates the alignment of the element's checkbox.

Type	Description
<a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that specifies the alignment of the checkbox.

By default, the CheckBoxAlign property is exMiddleLeft. Use the CheckBoxAlign property to align the element's checkbox. The [CheckElement](#) event notifies your application once the user clicks the element's check-box. The [ShowCheckBox](#) property specifies whether the element displays the check-box. The [Checked](#) property indicates whether the element is checked or unchecked. The [Background](#)( exCheckBoxState0) Specifies the visual appearance for the check box in 0 state. The [Background](#)(exCheckBoxState1) Specifies the visual appearance for the check box in 1 state.

# property Element.Checked as CheckStateEnum

Gets or sets a value that indicates the element's check-box state.

Type	Description
<a href="#">CheckStateEnum</a>	A CheckStateEnum expression that specifies the state of the element's check box.

By default, the Checked property is 0 ( unchecked ). The Checked property specifies the state of the element's check box. Use the [ShowCheckBox](#) property to show or hide the element's checkbox. The [CheckElement](#) event occurs once the state of the element's checkbox is changed. Use the [CheckBoxAlign](#) property to align the element's checkbox. The [Background](#)( exCheckBoxState0) Specifies the visual appearance for the check box in 0 state. The [Background](#)(exCheckBoxState1) Specifies the visual appearance for the check box in 1 state.

# property Element.ChildCount as Long

Counts the number of child elements.

Type	Description
Long	A Long expression that indicates the count of child elements.

The ChildCount property counts the number of child elements. The [VisibleChildCount](#) property specifies the list of visible children. The [Children](#) property specifies the list of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime. The [ParentChangeEvent](#) event occurs when the element's parent is changed.

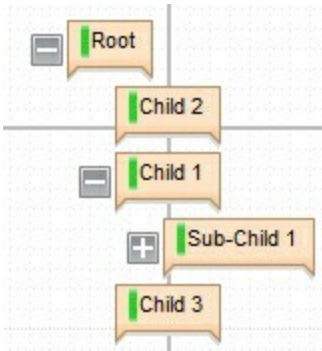
# property Element.ChildPosition as Long

Specifies the position of the element while it is a child element.

Type	Description
Long	A Long expression that specifies the position of the element in the parent's children collection.

The ChildPosition property indicates the position of the element in the parent's children collection. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime. The [ParentChangeEvent](#) event occurs when the element's parent is changed. The [Expanded](#) property expands or collapses a node.

The following screen shot shows the elements arranged as a tree:





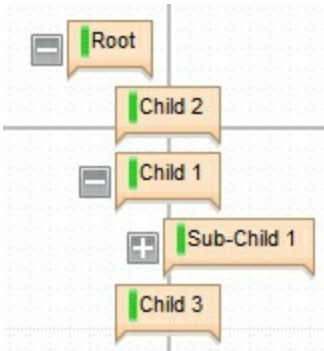
# property Element.Children as Variant

Returns a safe array of child elements.

Type	Description
Variant	A Safe-Array of elements indicating the list of child elements. You can use the for-each statement to enumerate the child elements of specified node.

The Children property specifies the list of child elements. The [VisibleChildren](#) property specifies the list of visible child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime. The [ParentChangeEvent](#) event occurs when the element's parent is changed.

The following screen shot shows the elements arranged as a tree:



# property Element.ClientPadding(Edge as PaddingEdgeEnum) as Long

Returns or sets a value that indicates the padding of the element's client.

Type	Description
Edge as <a href="#">PaddingEdgeEnum</a>	A PaddingEdgeEnum expression that specifies the edge to be changed.
Long	A long expression that specifies the padding.

The ClientPadding property specifies the padding while the element's [Type](#) property is exElementHostWindow or exElementHostControl. The ClientPadding property has no effect if the element's Type property is exElementHostDefault ( by default ). The [Padding](#) property specifies the body/background padding if the element's Type property is exElementHostDefault ( by default ). The [BorderPadding](#) property specifies the padding to be applied on borders, to define the position of the status and the body parts of the element. The [StatusPadding](#) property specifies the status padding.

# property Element.Control as String

Specifies the identifier of the inner control hosted by the current element.

Type	Description
String	A string expression that can be formatted as follows: a prog ID, a CLSID, a URL, a reference to an Active document , a fragment of HTML.

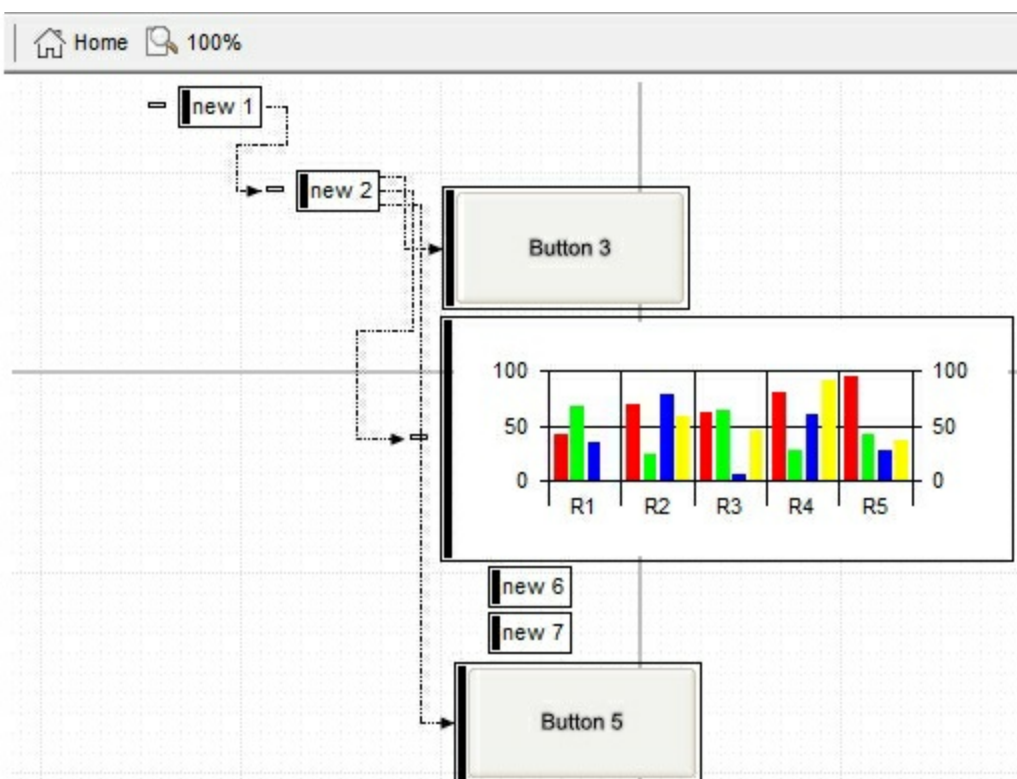
The control supports ActiveX hosting, or in other words, any element can host another inside controls. The Control property has effect only if the element's [Type](#) property is set on exElementHostControl. If you insert a runtime-licensed control you must specify the [License](#) property before calling the Control property. Use the [ElementFormat](#) property to specify the area where the inner control is displayed. The [Object](#) property returns a reference to newly created control. The control fires the [OLEEvent](#) event if an inside ActiveX control fires an event. Use the [InsertControl](#) property to add a new Element object with the [Type](#) set on exElementHostControl, that hosts an ActiveX control.

The Control property must be formatted in one of the following ways:

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

**The look and feel of the inner ActiveX control depends on the identifier you are using, and the version of the library that implements the ActiveX control, so you need to consult the documentation of the inner ActiveX control you are inserting inside the exSurface control.** The [License](#) property indicates a string expression that indicates the runtime license key for the component being inserted, if required. Only, the vendor of the component you are going to use is able to give you such of runtime license, so please contact the control's vendor for such of key. Your development license key is not compatible with the runtime license key, so it can't be used here.

The following screen shot shows the surface with different inner controls:



The following VB sample creates an element that hosts the [Exontrol.Button](#) control:

```
Private Sub Surface1_CreateElement(ByVal Element As EXSURFACELibCtl.IElement)
    With Element
        .Type = exElementHostControl
        .ElementFormat = ""client""
        .Control = "Exontrol.Button"
        With .Object
            .Caption = "<sha ;;0>Button " & Surface1.Elements.Count
        End With
    End With
End Sub
```

The samples changes the element's Type to exElementHostControl, and after that specifies the Control property to create the desired inner control.

The following samples shows how you can host the [Exontrol.Button](#) on the surface.

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

```
With Surface1
    With .Elements
        With .Add("ActiveX")
            .Type = 2
        End With
    End With
End With
```

```
.ElementFormat = ""check":18,"client""
.ShowCheckBox = True
.Control = "Exontrol.Button"
.Object.Caption = "<sha ;;0>button"
.Height = 32
.Width = 128
End With
End With
End With
```

## VB6

```
With Surface1
  With .Elements
    With .Add("ActiveX")
      .Type = exElementHostControl
      .ElementFormat = ""check":18,"client""
      .ShowCheckBox = True
      .Control = "Exontrol.Button"
      .Object.Caption = "<sha ;;0>button"
      .Height = 32
      .Width = 128
    End With
  End With
End With
End With
```

## VB.NET

```
With Exsurface1
  With .Elements
    With .Add("ActiveX")
      .Type = exontrol.EXSURFACELib.ElementHostTypeEnum.exElementHostControl
      .ElementFormat = ""check":18,"client""
      .ShowCheckBox = True
      .Control = "Exontrol.Button"
      .Object.Caption = "<sha ;;0>button"
      .Height = 32
      .Width = 128
    End With
  End With
End With
```

End With  
End With  
End With

## VB.NET for /COM

```
With AxSurface1
  With .Elements
    With .Add("ActiveX")
      .Type = EXSURFACELib.ElementHostTypeEnum.exElementHostControl
      .ElementFormat = """"check""":18, ""client""""
      .ShowCheckBox = True
      .Control = "Exontrol.Button"
      .Object.Caption = "<sha ;;0>button"
      .Height = 32
      .Width = 128
    End With
  End With
End With
```

## C++

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

  #import <ExSurface.dll>
  using namespace EXSURFACELib;
*/
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
>GetControlUnknown();
EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();
EXSURFACELib::IElementPtr var_Element = var_Elements-
>Add("ActiveX",vtMissing,vtMissing,vtMissing,vtMissing,vtMissing);
var_Element->PutType(EXSURFACELib::exElementHostControl);
var_Element->PutElementFormat(L""check"":18,\"client\");
var_Element->PutShowCheckBox(VARIANT_TRUE);
```

```
var_Element->PutControl(L"Exontrol.Button");
```

```
/*
```

Copy and paste the following directives to your header file as it defines the namespace 'EXBUTTONLib' for the library: 'ExButton 1.0 Control Library'

```
#import <ExButton.dll>
```

```
using namespace EXBUTTONLib;
```

```
*/
```

```
((EXBUTTONLib::IButtonPtr)(var_Element->GetObject()))->PutCaption(L"<sha ;;0>button");
```

```
var_Element->PutHeight(32);
```

```
var_Element->PutWidth(128);
```

## C++ Builder

```
Exsurfacelib_tlb::IElementsPtr var_Elements = Surface1->Elements;
```

```
Exsurfacelib_tlb::IElementPtr var_Element = var_Elements->Add(TVariant("ActiveX"),TNoParam(),TNoParam(),TNoParam(),TNoParam(),TNoParam()
```

```
var_Element->Type =
```

```
Exsurfacelib_tlb::ElementHostTypeEnum::exElementHostControl;
```

```
var_Element->ElementFormat = L"\check\":18,\"client\"";
```

```
var_Element->ShowCheckBox = true;
```

```
var_Element->Control = L"Exontrol.Button";
```

```
(IDispatch*)var_Element->Object->Caption = L"<sha ;;0>button";
```

```
var_Element->Height = 32;
```

```
var_Element->Width = 128;
```

## C#

```
exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;
```

```
exontrol.EXSURFACELib.Element var_Element =
```

```
var_Elements.Add("ActiveX",null,null,null,null,null);
```

```
var_Element.Type =
```

```
exontrol.EXSURFACELib.ElementHostTypeEnum.exElementHostControl;
```

```

var_Element.ElementFormat = "\"check\":18,\"client\"";
var_Element.ShowCheckBox = true;
var_Element.Control = "Exontrol.Button";
// Add 'ExButton 1.0 Control Library' reference to your project.
(var_Element.Object as exontrol.EXBUTTONLib.exbutton).Caption = "<sha
;;0>button";
var_Element.Height = 32;
var_Element.Width = 128;

```

## JavaScript

```

<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
  var var_Elements = Surface1.Elements;
  var var_Element = var_Elements.Add("ActiveX",null,null,null,null,null);
  var_Element.Type = 2;
  var_Element.ElementFormat = "\"check\":18,\"client\"";
  var_Element.ShowCheckBox = true;
  var_Element.Control = "Exontrol.Button";
  var_Element.Object.Caption = "<sha ;;0>button";
  var_Element.Height = 32;
  var_Element.Width = 128;
</SCRIPT>

```

## C# for /COM

```

EXSURFACELib.Elements var_Elements = axSurface1.Elements;
EXSURFACELib.Element var_Element =
var_Elements.Add("ActiveX",null,null,null,null,null);
var_Element.Type =
EXSURFACELib.ElementHostTypeEnum.exElementHostControl;
var_Element.ElementFormat = "\"check\":18,\"client\"";
var_Element.ShowCheckBox = true;
var_Element.Control = "Exontrol.Button";
// Add 'ExButton 1.0 Control Library' reference to your project.

```



```
(var_Element.Object as EXBUTTONLib.Button).Caption = "<sha ;;0>button";  
var_Element.Height = 32;  
var_Element.Width = 128;
```

## X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Element,com_Elements,com_Object;  
    anytype var_Element,var_Elements,var_Object;  
    ;  
  
    super();  
  
    var_Elements = exsurface1.Elements(); com_Elements = var_Elements;  
    var_Element = com_Elements.Add("ActiveX"); com_Element = var_Element;  
    com_Element.Type(2/*exElementHostControl*/);  
    com_Element.ElementFormat("\check\":18,\"client\");  
    com_Element.ShowCheckBox(true);  
    com_Element.Control("Exontrol.Button");  
    var_Object = COM::createFromObject(com_Element.Object()); com_Object =  
var_Object;  
    com_Object.Caption("<sha ;;0>button");  
    com_Element.Height(32);  
    com_Element.Width(128);  
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do  
begin  
    with Elements do  
    begin  
        with Add('ActiveX',Nil,Nil,Nil,Nil,Nil) do  
        begin  
            Type := EXSURFACELib.ElementHostTypeEnum.exElementHostControl;  
            ElementFormat := "check":18,"client";  
        end  
    end  
end
```

```

ShowCheckBox := True;
Control := 'Exontrol.Button';
(Object as EXBUTTONLib.Button).Caption := '<sha ;;0>button';
Height := 32;
Width := 128;
end;
end;
end

```

## Delphi (standard)

```

with Surface1 do
begin
  with Elements do
  begin
    with Add('ActiveX',Null,Null,Null,Null,Null) do
    begin
      Type := EXSURFACELib_TLB.exElementHostControl;
      ElementFormat := '"check":18,"client"';
      ShowCheckBox := True;
      Control := 'Exontrol.Button';
      (IUnknown(Object) as EXBUTTONLib_TLB.Button).Caption := '<sha ;;0>button';
      Height := 32;
      Width := 128;
    end;
  end;
end
end

```

## VFP

```

with thisform.Surface1
  with .Elements
    with .Add("ActiveX")
      .Type = 2
      .ElementFormat =
        "" + chr(34) + "check" + chr(34) + ":18," + chr(34) + "client" + chr(34) + ""
      .ShowCheckBox = .T.
      .Control = "Exontrol.Button"
    endwith
  endwith
endwith

```

```
.Object.Caption = "<sha ;;0>button"  
.Height = 32  
.Width = 128  
endwith  
endwith  
endwith
```

## dBASE Plus

```
local oSurface,var_Element,var_Elements  
  
oSurface = form.Activex1.nativeObject  
var_Elements = oSurface.Elements  
var_Element = var_Elements.Add("ActiveX")  
var_Element.Type = 2  
var_Element.ElementFormat = "" + [""] + "check" + [""] + ":18," + [""] + "client" +  
[""] + ""  
var_Element.ShowCheckBox = true  
var_Element.Control = "Exontrol.Button"  
var_Element.Object.Caption = "<sha ;;0>button"  
var_Element.Height = 32  
var_Element.Width = 128
```

## XBasic (Alpha Five)

```
Dim oSurface as P  
Dim var_Element as P  
Dim var_Elements as P  
  
oSurface = topparent:CONTROL_ACTIVEX1.activex  
var_Elements = oSurface.Elements  
var_Element = var_Elements.Add("ActiveX")  
var_Element.Type = 2  
var_Element.ElementFormat = "\"check\"":18,\"client\""  
var_Element.ShowCheckBox = .t.  
var_Element.Control = "Exontrol.Button"  
var_Element.Object.Caption = "<sha ;;0>button"
```

```
var_Element.Height = 32
var_Element.Width = 128
```

## Visual Objects

```
local var_Element as IElement
local var_Elements as IElements

var_Elements := oDCOCX_Exontrol1:Elements
  var_Element := var_Elements:Add("ActiveX",nil,nil,nil,nil)
    var_Element:Type := exElementHostControl
    var_Element:ElementFormat := "" + CHR(34) + "check" + CHR(34) + ":18," +
CHR(34) + "client" + CHR(34) + ""
    var_Element:ShowCheckBox := true
    var_Element:Control := "Exontrol.Button"
    // Generate Source for 'ExButton 1.0 Control Library' server from
Tools\Automation Server...
    IButton{var_Element:Object}:Caption := "<sha ;;0>button"
    var_Element.Height := 32
    var_Element.Width := 128
```

## PowerBuilder

```
OleObject oSurface,var_Element,var_Elements

oSurface = ole_1.Object
var_Elements = oSurface.Elements
  var_Element = var_Elements.Add("ActiveX")
    var_Element.Type = 2
    var_Element.ElementFormat = "" + CHAR(34) + "check" + CHAR(34) + ":18," +
CHAR(34) + "client" + CHAR(34) + ""
    var_Element.ShowCheckBox = true
    var_Element.Control = "Exontrol.Button"
    var_Element.Object.Caption = "<sha ;;0>button"
    var_Element.Height = 32
    var_Element.Width = 128
```

# Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Variant voElements
  Get ComElements to voElements
  Handle hoElements
  Get Create (RefClass(cComElements)) to hoElements
  Set pvComObject of hoElements to voElements
    Variant voElement
    Get ComAdd of hoElements "ActiveX" Nothing Nothing Nothing Nothing
  Nothing to voElement
  Handle hoElement
  Get Create (RefClass(cComElement)) to hoElement
  Set pvComObject of hoElement to voElement
    Set ComType of hoElement to OLEexElementHostControl
    Set ComElementFormat of hoElement to ""check":18,"client""
    Set ComShowCheckBox of hoElement to True
    Set ComControl of hoElement to "Exontrol.Button"
    Variant voButton
    Get ComObject of hoElement to voButton
    Handle hoButton
    Get Create (RefClass(cComButton)) to hoButton
    Set pvComObject of hoButton to voButton
      Set ComCaption of hoButton to "<sha ;;0>button"
    Send Destroy to hoButton
    Set ComHeight of hoElement to 32
    Set ComWidth of hoElement to 128
  Send Destroy to hoElement
  Send Destroy to hoElements
End_Procedure
```

# XBase++

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

## PROCEDURE Main

LOCAL oForm

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

LOCAL oElement

LOCAL oElements

LOCAL oSurface

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

oForm:drawingArea:clipChildren := .T.

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

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

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

oSurface:CLSID := "Exontrol.Surface.1" /\*{AC1DF7F4-0919-4364-8167-2F9B5155EA4B}\*/

oSurface:create(„ {10,60},{610,370} )

oElements := oSurface:Elements()

oElement := oElements:Add("ActiveX")

oElement:**Type** := 2/\*exElementHostControl\*/

oElement:ElementFormat := "" + CHR(34) + "check" + CHR(34) + ":18," +  
CHR(34) + "client" + CHR(34) + ""

oElement:ShowCheckBox := .T.

oElement:**Control** := "Exontrol.Button"

oElement:Object():Caption := "<sha ;;0>button"

oElement:Height := 32

oElement:Width := 128

oForm:Show()

DO WHILE nEvent != xbeP\_Quit

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

oXbp:handleEvent( nEvent, mp1, mp2 )

ENDDO

RETURN



# method Element.Edit (Part as EditEnum, [Options as Variant])

Edits the element.

Type	Description
Part as <a href="#">EditEnum</a>	An EditEnum expression that specifies the caption to be edited such as <a href="#">Caption</a> or <a href="#">ExtraCaption</a> .
Options as Variant	A String expression that specifies the options to show the inline editing. The Options are separated by comma character (,).
Return	Description
Variant	A Long expression that specifies whether the user presses the ENTER while inline editing. The Valid values are 0 if user presses the ESC key during inline editing, -1 if the user presses the ENTER key, or 1 if the user closes the form.

Use the Edit method to inline edit the element's caption or extra-caption. The [Caption](#) property indicates the element's HTML Caption. The [ExtraCaption](#) property indicates the element's HTML Extra-Caption. The [CaptionSingleLine](#) property specifies whether the element's caption displays single or multiple lines. The [ExtraCaptionSingleLine](#) property specifies whether the element's extra-caption displays single or multiple lines. The control fires the [LayoutStartChanging](#)(exEditObject) event when the Edit method starts, and the [LayoutEndChanging](#)(exEditObject) event when the Edit method ends.

By default, the edit's background is not changed so the inline editing is transparent. Use the [Background](#)(exEditBackColor) to specify the inline editing's background color. Use the [Background](#)(exEditForeColor) to specify the inline editing's foreground color. Use the [Background](#)(exEditSelBackColor) to specify the inline editing's selection background color. Use the [Background](#)(exEditSelForeColor) to specify the inline editing's selection foreground color.

The Options parameter can include one or more of the following options:

- **multiline**, if present, the inline editing allows editing multiple lines, so ENTER key inserts a new line, while the CTRL + ENTER validates or ends editing the field.
- **wordwrap**, if present, the inline editor word wraps the text. The **multiline** should be set too, so the edit can display multiple lines
- **tab**, if present, the TAB key inserts a TAB character to inline caption
- **nocolor**, specifies that no-colors are applied to inline editing.



# property Element.ElementFormat as String

Specifies the way the control shows the parts of the element.

Type	Description
String	A String format that specified the <a href="#">CRD</a> format to arrange the parts of the element.

By default, the ElementFormat property is empty. Use the ElementFormat property to arrange in a different way the parts of the element. If the ElementFormat property is empty, the format of the elements is specified by the control's [ElementFormat](#) property. In other words, all elements can be formatted the same way using the control's [ElementFormat](#) property or separate way using the Element's ElementFormat property. The [DrawPartsOrder](#) property defines the order of the parts the elements display. The DrawPartsOrder property has no effect if the ElementFormat property is set.

The know parts of the element are:

- **check**, specifies the part where the element's checkbox is displayed. The [ShowCheckBox](#) property specifies whether the element's checkbox is shown or hidden. *The [CheckBoxAlign](#) property specifies the alignment of the checkbox relative to the "check" part of the element.*
- **caption**, specifies the part where the element's caption is displayed. The [Caption](#) property specifies the element's HTML caption. *The [CaptionAlign](#) property specifies the alignment of the caption relative to the "caption" part of the element.*
- **extracaption**, specifies the part where the element's extra-caption is displayed. The [ExtraCaption](#) property specifies the element's HTML extra-caption. *The [ExtraCaptionAlign](#) property specifies the alignment of the extra-caption relative to the "extracaption" part of the element.*
- **picture**, specifies the part where the element's pictures are displayed. The [Pictures](#) property specifies the element's pictures. *The [PicturesAlign](#) property specifies the alignment of the pictures relative to the "picture" part of the element.*
- **extrapicture**, specifies the part where the element's extra-pictures are displayed. The [ExtraPictures](#) property specifies the element's extra-pictures. *The [ExtraPicturesAlign](#) property specifies the alignment of the extra-pictures relative to the "extrapicture" part of the element.*
- **client**, specifies the part of the element where the inside ActiveX is displayed. The [Control](#) property indicates the inside ActiveX that hosted by the element.

The parts of the elements must be included between "" in order to be recognized by the [CRD](#) format.

For instance:

- "check" indicates that just the element's checkbox is displayed, so no matter if other are set the element displays just the checkbox ( if [ShowCheckBox](#) property is True ) and the element's Caption property.
- "check,caption" indicates that just the element's checkbox and caption are displayed, so no matter if other are set the element displays just the checkbox ( if [ShowCheckBox](#) property is True ) and the element's [Caption](#) property.
- "client" specifies that the whole element displays just the client part, so the inside Active control will use the entire background to display the inside ActiveX control. This indicates, that no caption, check or any other part is displayed on the element.
- "check":18,"client", displays the element's checkbox aligned to the left on a 18-pixels wide, and displays the client on the rest of the element.
- "18;"caption"/"client"", allows the element's [Caption](#) and the [Control](#) to be displayed.

The following samples show how you can display the element's checkbox next to the Command button.

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

```
With Surface1
  With .Elements
    With .InsertControl("Forms.CommandButton.1")
      .ElementFormat = """"check""":18,""client""""
      .Object.Caption = "command"
      .ShowCheckBox = True
      .Height = 48
      .Width = 128
    End With
  End With
End With
```

### VB6

```
With Surface1
  With .Elements
    With .InsertControl("Forms.CommandButton.1")
      .ElementFormat = """"check""":18,""client""""
      .Object.Caption = "command"
      .ShowCheckBox = True
      .Height = 48
      .Width = 128
    End With
  End With
End With
```

```
End With
End With
End With
```

## VB.NET

```
With Exsurface1
  With .Elements
    With .InsertControl("Forms.CommandButton.1")
      .ElementFormat = """"check""":18,""client""""
      .Object.Caption = "command"
      .ShowCheckBox = True
      .Height = 48
      .Width = 128
    End With
  End With
End With
```

## VB.NET for /COM

```
With AxSurface1
  With .Elements
    With .InsertControl("Forms.CommandButton.1")
      .ElementFormat = """"check""":18,""client""""
      .Object.Caption = "command"
      .ShowCheckBox = True
      .Height = 48
      .Width = 128
    End With
  End With
End With
```

## C++

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

```

#import <ExSurface.dll>
using namespace EXSURFACELib;
*/
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
>GetControlUnknown();
EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();
EXSURFACELib::IElementPtr var_Element = var_Elements-
> InsertControl("Forms.CommandButton.1",vtMissing,vtMissing,vtMissing);
var_Element->PutElementFormat(L"\check\":18,\"client\");
/*

```

Copy and paste the following directives to your header file as it defines the namespace 'MSForms' for the library: 'Microsoft Forms 2.0 Object Library'

```

#import <FM20.DLL>
*/
((MSForms::ICommandButtonPtr)(var_Element->GetObject()))-
>PutCaption(L"command");
var_Element->PutShowCheckBox(VARIANT_TRUE);
var_Element->PutHeight(48);
var_Element->PutWidth(128);

```

## C++ Builder

```

Exsurfacelib_tlb::IElementsPtr var_Elements = Surface1->Elements;
Exsurfacelib_tlb::IElementPtr var_Element = var_Elements-
> InsertControl(TVariant("Forms.CommandButton.1"),TNoParam(),TNoParam(),TNoParam());

var_Element->ElementFormat = L"\check\":18,\"client\");
(IDispatch*)var_Element->Object->Caption = L"command";
var_Element->ShowCheckBox = true;
var_Element->Height = 48;
var_Element->Width = 128;

```

## C#

```

exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;
    exontrol.EXSURFACELib.Element var_Element =
var_Elements.InsertControl("Forms.CommandButton.1",null,null,null);
    var_Element.ElementFormat = "\"check\":18,\"client\"";
// Add 'Microsoft Forms 2.0 Object Library' reference to your project.
    (var_Element.Object as MSForms.CommandButton).Caption = "command";
    var_Element.ShowCheckBox = true;
    var_Element.Height = 48;
    var_Element.Width = 128;

```

## JavaScript

```

<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
    var var_Elements = Surface1.Elements;
    var var_Element =
var_Elements.InsertControl("Forms.CommandButton.1",null,null,null);
    var_Element.ElementFormat = "\"check\":18,\"client\"";
    var_Element.Object.Caption = "command";
    var_Element.ShowCheckBox = true;
    var_Element.Height = 48;
    var_Element.Width = 128;
</SCRIPT>

```

## C# for /COM

```

EXSURFACELib.Elements var_Elements = axSurface1.Elements;
    EXSURFACELib.Element var_Element =
var_Elements.InsertControl("Forms.CommandButton.1",null,null,null);
    var_Element.ElementFormat = "\"check\":18,\"client\"";
// Add 'Microsoft Forms 2.0 Object Library' reference to your project.
    (var_Element.Object as MSForms.CommandButton).Caption = "command";
    var_Element.ShowCheckBox = true;
    var_Element.Height = 48;
    var_Element.Width = 128;

```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Element,com_Elements,com_Object;
    anytype var_Element,var_Elements,var_Object;
    ;

    super();

    var_Elements = exsurface1.Elements(); com_Elements = var_Elements;
    var_Element = com_Elements.InsertControl("Forms.CommandButton.1");
    com_Element = var_Element;
    com_Element.ElementFormat("\check\:18,\client\");
    var_Object = COM::createFromObject(com_Element.Object()); com_Object =
    var_Object;
    com_Object.Caption("command");
    com_Element.ShowCheckBox(true);
    com_Element.Height(48);
    com_Element.Width(128);
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do
begin
    with Elements do
    begin
        with InsertControl('Forms.CommandButton.1',Nil,Nil,Nil) do
        begin
            ElementFormat := '"check":18,"client"';
            (Object as MSForms.CommandButton).Caption := 'command';
            ShowCheckBox := True;
            Height := 48;
            Width := 128;
        end;
    end;
end;
```

```
end;  
end
```

## Delphi (standard)

```
with Surface1 do  
begin  
  with Elements do  
  begin  
    with InsertControl('Forms.CommandButton.1',Null,Null,Null) do  
    begin  
      ElementFormat := "'check':18,'client';  
(IUnknown(Object) as MSForms_TLB.CommandButton).Caption := 'command';  
      ShowCheckBox := True;  
      Height := 48;  
      Width := 128;  
    end;  
  end;  
end
```

## VFP

```
with thisform.Surface1  
  with .Elements  
    with InsertControl("Forms.CommandButton.1")  
      .ElementFormat =  
      "" + chr(34) + "check" + chr(34) + ":18," + chr(34) + "client" + chr(34) + ""  
      .Object.Caption = "command"  
      .ShowCheckBox = .T.  
      .Height = 48  
      .Width = 128  
    endwith  
  endwith  
endwith
```

## dBASE Plus

```
local oSurface,var_Element,var_Elements
```

```

oSurface = form.ActiveX1.nativeObject
var_Elements = oSurface.Elements
  var_Element = var_Elements.InsertControl("Forms.CommandButton.1")
    var_Element.ElementFormat = "" + [""] + "check" + [""] + ":18," + [""] + "client" +
[""] + ""
    var_Element.Object.Caption = "command"
    var_Element.ShowCheckBox = true
    var_Element.Height = 48
    var_Element.Width = 128

```

## XBasic (Alpha Five)

```

Dim oSurface as P
Dim var_Element as P
Dim var_Elements as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
var_Elements = oSurface.Elements
  var_Element = var_Elements.InsertControl("Forms.CommandButton.1")
    var_Element.ElementFormat = "\"check\"":18,\"client\"""
    var_Element.Object.Caption = "command"
    var_Element.ShowCheckBox = .t.
    var_Element.Height = 48
    var_Element.Width = 128

```

## Visual Objects

```

local var_Element as IElement
local var_Elements as IElements

var_Elements := oDCOCX_Exontrol1:Elements
  var_Element := var_Elements:InsertControl("Forms.CommandButton.1",nil,nil,nil)
    var_Element:ElementFormat := "" + CHR(34) + "check" + CHR(34) + ":18," +
CHR(34) + "client" + CHR(34) + ""
    // Generate Source for 'Microsoft Forms 2.0 Object Library' server from

```



## Tools\Automation Server...

```
ICommandButton{var_Element:Object}:Caption := "command"  
var_Element.ShowCheckBox := true  
var_Element.Height := 48  
var_Element.Width := 128
```

## PowerBuilder

```
OleObject oSurface,var_Element,var_Elements
```

```
oSurface = ole_1.Object
```

```
var_Elements = oSurface.Elements
```

```
var_Element = var_Elements.InsertControl("Forms.CommandButton.1")
```

```
var_Element.ElementFormat = "" + CHAR(34) + "check" + CHAR(34) + ":18," +  
CHAR(34) + "client" + CHAR(34) + ""
```

```
var_Element.Object.Caption = "command"
```

```
var_Element.ShowCheckBox = true
```

```
var_Element.Height = 48
```

```
var_Element.Width = 128
```

## Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Variant voElements
```

```
Get ComElements to voElements
```

```
Handle hoElements
```

```
Get Create (RefClass(cComElements)) to hoElements
```

```
Set pvComObject of hoElements to voElements
```

```
Variant voElement
```

```
Get ComInsertControl of hoElements "Forms.CommandButton.1" Nothing
```

```
Nothing Nothing to voElement
```

```
Handle hoElement
```

```
Get Create (RefClass(cComElement)) to hoElement
```

```
Set pvComObject of hoElement to voElement
```

```
Set ComElementFormat of hoElement to ""check":18,"client""
```

```

Variant voCommandButton
Get ComObject of hoElement to voCommandButton
Handle hoCommandButton
Get Create (RefClass(cComCommandButton)) to hoCommandButton
Set pvComObject of hoCommandButton to voCommandButton
    Set ComCaption of hoCommandButton to "command"
Send Destroy to hoCommandButton
Set ComShowCheckBox of hoElement to True
Set ComHeight of hoElement to 48
Set ComWidth of hoElement to 128
Send Destroy to hoElement
Send Destroy to hoElements
End_Procedure

```

## XBase++

```

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

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oElement
    LOCAL oElements
    LOCAL oSurface

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oSurface := XbpActiveXControl():new( oForm:drawingArea )
    oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
    oSurface:create(,, {10,60},{610,370} )

    oElements := oSurface:Elements()

```

```
oElement := oElements:InsertControl("Forms.CommandButton.1")
oElement:ElementFormat := "" + CHR(34) + "check" + CHR(34) + ":18," +
CHR(34) + "client" + CHR(34) + ""
oElement:Object():Caption := "command"
oElement:ShowCheckBox := .T.
oElement:Height := 48
oElement:Width := 128
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

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

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

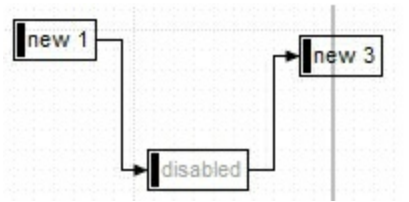
# property Element.Enabled as Boolean

Gets or sets a value that indicates whether the element is enabled or disabled.

Type	Description
Boolean	A Boolean expression that specifies whether the element is enabled or disabled.

By default, the Enabled property is True. Use the Enabled property to show the element as disabled ( grayed ). The [Selectable](#) property specifies whether the user can select the element at runtime. The [Resizable](#) property specifies whether the element can be resized at runtime. Use the [Enabled](#) property of the control to disable the entire surface.

The following screen shot shows a few elements, and one disabled element:



## method `Element.EndUpdateElement (StartUpdateElement as Long)`

Adds programmatically updated properties of the element to undo/redo queue.

Type	Description
StartUpdateElement as Long	A long expression that specifies the handle being returned by the <a href="#">StartUpdateElement</a> property

The [StartUpdateElement](#)/EndUpdateElement methods record and add changes of the current element to the control's Undo/Redo queue. You can use the [StartBlockUndoRedo](#) / [EndBlockUndoRedo](#) methods to group multiple Undo/Redo operations into a single-block. The [AllowUndoRedo](#) property specifies whether the control supports undo/redo operations for objects (elements, links, ...). No entry is added to the Undo/Redo queue if no property is changed for the current element. Each call of the [StartUpdateElement](#) must be succeeded by a EndUpdateElement call. The [UndoListAction](#) property lists the Undo actions that can be performed in the chart. The [RedoListAction](#) property lists the Redo actions that can be performed in the chart.

The [StartUpdateElement](#)/EndUpdateElement methods can record changes for the following properties only:

- [Caption](#), gets or sets a value that indicates the HTML caption to be displayed on the element
- [Parent](#), defines the element's parent (when the element is part of a hierarchy)
- [ChildPosition](#), specifies the position of the element while it is a child element (when the element is part of a hierarchy)
- [ID](#), defines the element's unique identifier
- [AutoSize](#), specifies if the element computes its size automatically
- [X](#), specifies the element's x-position
- [Y](#), specifies the element's y-position
- [Width](#), specifies the width of the element
- [Height](#), specifies the height of the element
- [Visible](#), shows or hides the element
- [Enabled](#), enables or disables the element
- [BackColor](#), gets or sets a value that indicates the element's background color
- [ForeColor](#), gets or sets a value that indicates the element's foreground color
- [OverviewColor](#), gets or sets a value that indicates the element's overview color
- [BorderColor](#), gets or sets a value that indicates the element's border color
- [StatusColor](#), gets or sets a value that indicates the element's status color
- [Pictures](#), specifies the list of pictures to be displayed on the element
- [ExtraPictures](#), specifies the list of additional pictures to be displayed on the element
- [ExtraCaption](#), gets or sets a value that indicates additional HTML caption to be displayed on the element

- [Pattern](#), specifies the pattern to be shown on the element's background
- [UserData](#), associates any extra data associated with the element

The Undo/Redo records show as:

- "**UpdateElement**;ELEMENTID", indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / EndUpdateElement methods

within the [UndoListAction](#)/[RedoListAction](#) result

# method Element.EnsureVisible ()

Scrolls the surface to ensure that the current element fits the control's visible area.

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

The EnsureVisible method scrolls the surface to ensure that the current element fits the control's visible area. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position.

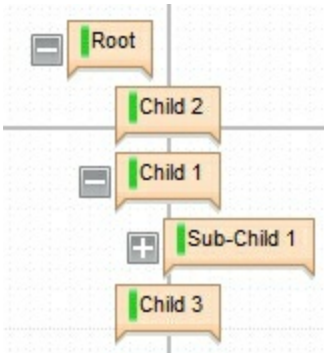
# property Element.Expanded as Boolean

Expands or collapses the element.

Type	Description
Boolean	A Boolean expression that specifies whether the element is expanded or collapsed.

The control displays a +/- expanding glyphs next to the parent elements that contain child elements or outgoing elements ([ExpandLinkedElements](#) property). The Expanded property specifies whether the element is expanded or collapsed. The [ExpandLinkedElements](#) property specifies whether the elements displays the expand/collapse glyphs when the element has outgoing elements ( the [OutgoingLinks](#) property specifies the links that starts from the element ). The [Background](#)(exTreeGlyphCollapsed) and [Background](#)(exTreeGlyphExpanded) specifies the visual appearance to show the glyph next to the collapsed/expanded element. The [Parent](#) property specifies the element's parent. The [Children](#) property specifies the list of child elements. The control fires the [ExpandElement](#) event when a node is collapsed or expanded. The [Add](#) method adds programmatically a link between two elements. Use the [Insert](#) method to insert programmatically a child element. Use the [ShowLinksOnCollapse](#) property to show the links between an element and collapsed elements. Use the [IndentX](#) / [IndentY](#) property to specify the indentation between child and parent elements.

The following screen shot shows the elements arranged as a tree:





# property Element.ExtraCaption as String

Gets or sets a value that indicates the extra HTML caption to be displayed on the element.

Type	Description
String	A String expression that defines the HTML extra-caption to be displayed on the element's background.

By default, the ExtraCaption property is empty. The ExtraCaption property defines the second or the extra caption to be displayed on the element's background. Use the [Caption](#) property to define the label or caption to be displayed on the element's background. The [ExtraCaptionAlign](#) property specifies the alignment of the extra-caption relative to the edges of the element. The [ExtraCaptionSingleLine](#) property specifies whether the element's caption is displayed on single or multiple lines. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

The ExtraCaption property supports the following HTML elements:

- **<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.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using <a ;exp=> or <a ;e64=> anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu</a>" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY string encodes the "<fgcolor 808080>show lines<a>-</a></fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to

decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "<font Tahoma;12>bit</font>" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "<font ;12>bit</font>" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the

picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.

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

gradient-center

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

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or **<fgcolor>** defines the color to show the inside text. The **<font>** HTML tag can be used to define the height of the font. For instance the "**<font ;31><sha>**shadow**</sha></font>**" generates the following picture:

shadow

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

outline anti-aliasing

# property Element.ExtraCaptionAlign as ContentAlignmentEnum

Indicates the alignment of the element's extra caption.

Type	Description
<a href="#">ContentAlignmentEnum</a>	<p>A ContentAlignmentEnum expression that specifies the extra-caption alignment relative to the element's edges. The ExtraCaptionAlign property supports additionally the following flag:</p> <ul style="list-style-type: none"><li>• exWidth (4), to distribute the text on the element's width</li></ul>

The [ExtraCaptionAlign](#) property specifies the alignment of the extra-caption relative to the edges of the element. The [ExtraCaptionSingleLine](#) property specifies whether the element's caption is displayed on single or multiple lines. The [ExtraCaption](#) property defines the second or the extra caption to be displayed on the element's background. Use the [Caption](#) property to define the label or caption to be displayed on the element's background. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

# property Element.ExtraCaptionSingleLine as CaptionSingleLineEnum

Specifies if the element's extra caption is displayed on single or multiple lines.

Type	Description
<a href="#">CaptionSingleLineEnum</a>	<p>A CaptionSingleLineEnum expression that specifies whether the element's extra caption is displayed on a single or multiple lines. The ExtraCaptionAlign property supports additionally the following flag:</p> <ul style="list-style-type: none"><li>• exJustify (4), distributes the text evenly between the margins</li></ul>

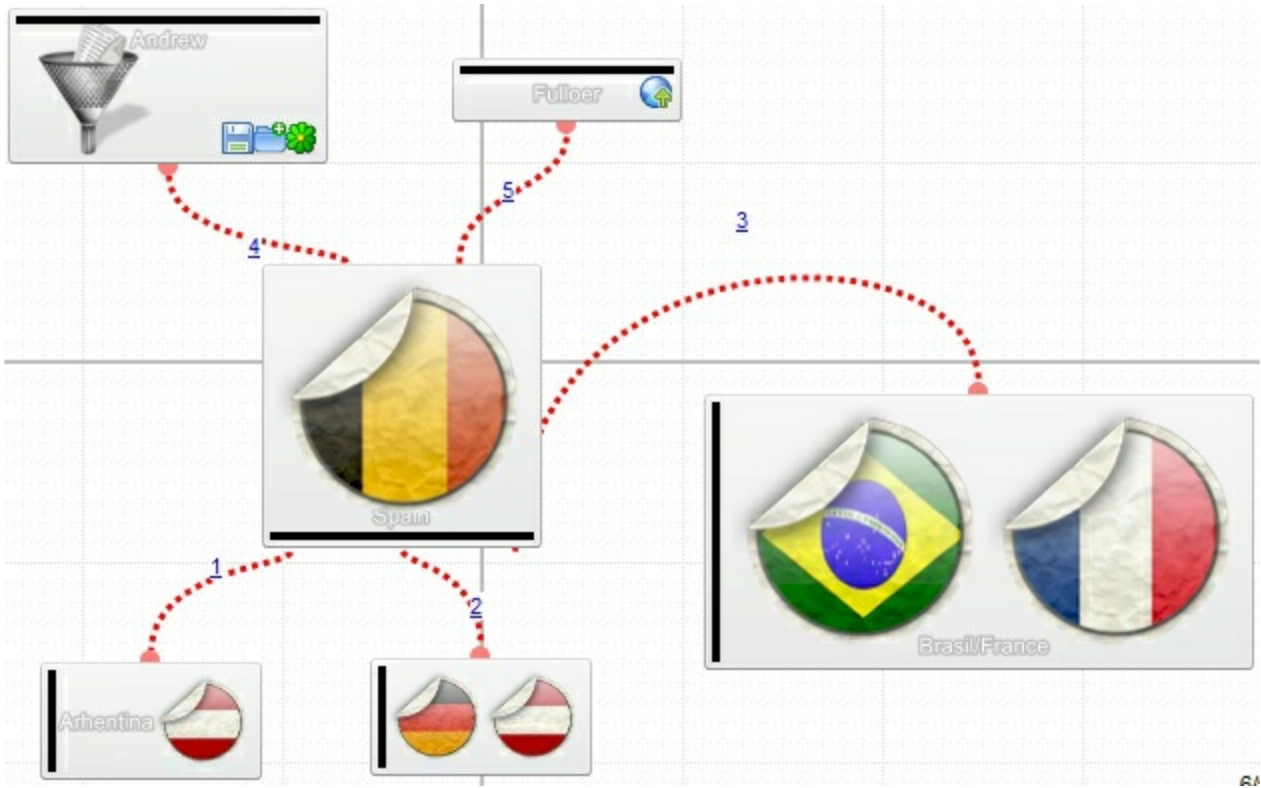
By default, the ExtraCaptionSingleLine property is exCaptionSingleLine. The ExtraCaptionSingleLine property specifies whether the element's caption is displayed on single or multiple lines. The [ExtraCaption](#) property defines the second or the extra caption to be displayed on the element's background. Use the [Caption](#) property to define the label or caption to be displayed on the element's background. The [ExtraCaptionAlign](#) property specifies the alignment of the extra-caption relative to the edges of the element. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

# property Element.ExtraPictures as String

Specifies the list of extra pictures to be displayed on the element.

Type	Description
String	A string expression that specifies the list of pictures to be shown on the element's body. The element's body can display one ore more pictures at the time, on different lines. For instance: "1,2/pic1" displays the 1 and 2 icons on the first line, while pic2 is displayed on the second line.

By default, the ExtraPictures property is "", which means that initially no extra-pictures are being displayed on the element. The [Pictures](#) or/and ExtraPictures property displays a collection of icons, pictures in the element's body. The Picture and ExtraPictures may display one or more pictures at the time. The , character indicates the separator of pictures in the same line, while the / character divides the lines to show the pictures. For instance, "1,2" displays icon with the index 1 and 2 on the same line, while the "1/2,pic1" displays the first icon on the first line, the second icon and the picture pic1 on the second line. The [Images](#) method loads icons to the control. The Images collection can display only 16x16 icons. The [HTMLPicture](#) assigns a key to a picture object. The [ShowHandCursorOn](#) property specifies whether the hand cursor is shown when hovering a picture on the element. The [HandCursorClick](#) event occurs once the user clicks a picture on the element ( [ShowHandCursorOn](#) property must include the exShowHandCursorPicture, exShowHandCursorIcon and exShowHandCursorExtraPictures ). The [ExtraPicturesAlign](#) property specifies the alignment of the extra-Pictures relative to the element.



The [Picture](#) property displays a picture on the element's background. The [PictureDisplay](#) property specifies the way the element's picture is displayed on the element's background. The [BackColor](#) property specifies the element's background color. [Background](#)(exElementBackColor) property specifies the default background color / visual appearance. The [ForeColor](#) property specifies the element's foreground color. The [Pattern](#) property defines the pattern to be shown on the control's background.



# property Element.ExtraPicturesAlign as ContentAlignmentEnum

Indicates the alignment of the element's extra picture.

Type	Description
<a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that specifies the extra-pictures alignment.

By default, the ExtraPicturesAlign property is exTopRight. The ExtraPicturesAlign property specifies the alignment of the extra-Pictures relative to the element. By default, the [ExtraPictures](#) property is "", which means that initially no extra-pictures are being displayed on the element. The [Pictures](#) or/and [ExtraPictures](#) property displays a collection of icons, pictures in the element's body. The Picture and ExtraPictures may display one or more pictures at the time. The , character indicates the separator of pictures in the same line, while the / character divides the lines to show the pictures. For instance, "1,2" displays icon with the index 1 and 2 on the same line, while the "1/2,pic1" displays the first icon on the first line, the second icon and the picture pic1 on the second line. The [Images](#) method loads icons to the control. The Images collection can display only 16x16 icons. The [HTMLPicture](#) assigns a key to a picture object. The [ShowHandCursorOn](#) property specifies whether the hand cursor is shown when hovering a picture on the element. The [HandCursorClick](#) event occurs once the user clicks a picture on the element ( [ShowHandCursorOn](#) property must include the exShowHandCursorPicture, exShowHandCursorIcon and exShowHandCursorExtraPictures ).

# property Element.FirstChild as Element

Gets the first child of the element.

Type	Description
<a href="#">Element</a>	An Element object that specifies the first child element of the current element.

The FirstChild property returns nothing if the current element contains no child elements. The FirstChild property returns the first child element. Use the FirstChild and [NextSiblingChild](#) properties to enumerate child elements one by one. The [LastChild](#) property indicates the last child element. Use the [LastChild](#) and [PrevSiblingChild](#) properties to backward enumerate all child elements. The [NextVisibleChild](#) property indicates the next visible element. The [PrevVisibleChild](#) property indicates the previously visible element. The [Children](#) property retrieves the child elements at once. The [ChildCount](#) property specifies the number of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime.

# property Element.ForeColor as Color

Gets or sets a value that indicates the element's foreground color.

Type	Description
Color	A Color expression that defines the element's foreground color.

By default, the ForeColor property is -1, which indicates that the element's foreground is defined by the [Background](#)(exElementForeColor). The ForeColor property specifies the element's foreground color. The [Pattern](#) property defines the pattern to be shown on the control's background. The [Padding](#) property defines the padding of the element. The [BorderColor](#) property specifies the color to show the border for a specific element. The [StatusColor](#) property specifies the color or the visual appearance to show the element's status part.

# property Element.Height as Long

Specifies the height of the element.

Type	Description
Long	A Long expression that specifies the height in pixels of the element.

The Height property specifies the height of the element. The [Width](#) property specifies the width of the element. Use the [Width](#) and Height properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the Height can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size.

# property Element.ID as Variant

Specifies the element's unique identifier.

Type	Description
Variant	A Long, String or Numeric expression that specifies the unique identifier of the Element.

The ID property is automatically assigned by the control once a new element is added. You can change the ID property to a different value, unless there is no other element with the same ID. In other words, the surface must contains elements with different IDs. The [CreateElement](#) event notifies your application once a new element is created. When elements are saved to an XML document using the [SaveXML](#), the ID will be as string once the [LoadXML](#) method is called ( the XML file is a TEXT file )

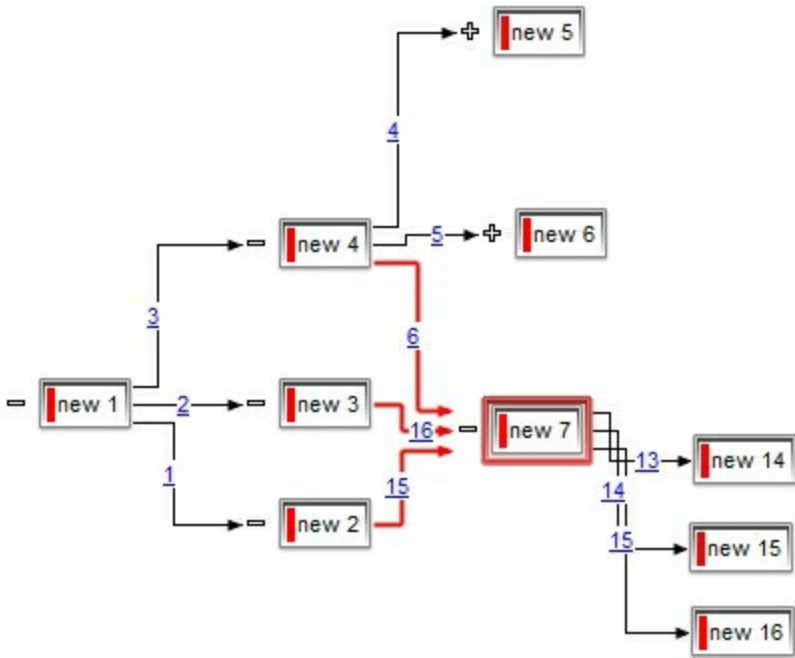
# property Element.IncomingLinks as Variant

Returns a safe array of incoming links.

Type	Description
Variant	A safe-array of <a href="#">Link</a> objects that specifies the links that ends to the current element. You can use the for-each statement to enumerate all incoming links.

The IncomingLinks property specifies the list of links that ends on the current element. The [OutgoingLinks](#) property specifies the list of links that starts from the current element. The [ElementFrom](#) property of the Link object indicates where the Link starts. The [ElementTo](#) property of the Link object indicates where the Link ends. Use the [ShowLinksColor](#)(exShowLinksEndTo)/[ShowLinksStyle](#)(exShowLinksEndTo)/[ShowLinksWidth](#)( properties to mark the incoming links of selected elements.

The following screen shot shows the incoming links (red ):



The following VB sample enumerates the incoming elements ( of selected elements ):

```
Private Sub Surface1_SelectionChanged()  
    With Surface1  
        Dim s As Variant  
        For Each s In .Selection  
            Debug.Print "Incomming Elements of " & s.ID & "are: "  
            With s  
                For Each i In .IncomingLinks
```

```
Debug.Print i.ElementFrom.ID
```

```
Next
```

```
End With
```

```
Next
```

```
End With
```

```
End Sub
```

# property Element.InflateSize as Long

Increases or decreases the width and height of the element.

Type	Description
Long	A Long expression that specifies the width and height of the element to be increased with when <a href="#">AutoSize</a> property is True.

By default, the The InflateSize property is 1 ( pixels). The InflateSize property indicates the size to be added to the default auto-size for increasing the size of the element. The [Padding](#), [BorderPadding](#) and [StatusPadding](#) properties specifies the padding to be applied on client, border and status parts of the element. The [Caption](#) property specifies the element's caption. The [ExtraCaption](#) property specifies the element's extra caption. The [ShowCheckBox](#) property indicates whether the element's checkbox is visible or hidden.



# property Element.LastChild as Element

Gets the last child of the element.

Type	Description
<a href="#">Element</a>	An Element object that specifies the last child element of the current element.

The LastChild property returns nothing if the current element contains no child elements. The LastChild property indicates the last child element. The [FirstChild](#) property returns the first child element. Use the FirstChild and [NextSiblingChild](#) properties to enumerate child elements one by one. Use the LastChild and [PrevSiblingChild](#) properties to backward enumerate all child elements. The [NextVisibleChild](#) property indicates the next visible element. The [PrevVisibleChild](#) property indicates the previously visible element. The [Children](#) property retrieves the child elements at once. The [ChildCount](#) property specifies the number of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime.

# property Element.Level as Long

Specifies the level of the element in a hierarchy.

Type	Description
Long	A Long expression that specifies the level of the element in the hierarchy. For instance, an element with no parent has the Level 0.

The Level property specifies the level in the hierarchy of the element. The [Parent](#) property specifies the parent of the element. The Level of the Element is defined as the Level of the Parent element + 1. The [ChildCount](#) property specifies the number of child elements. The [Children](#) property specifies the list of child elements. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime.

## property Element.License as String

Indicates the runtime license required to create the inner control.

Type	Description
String	A String expression that specifies the runtime-license.

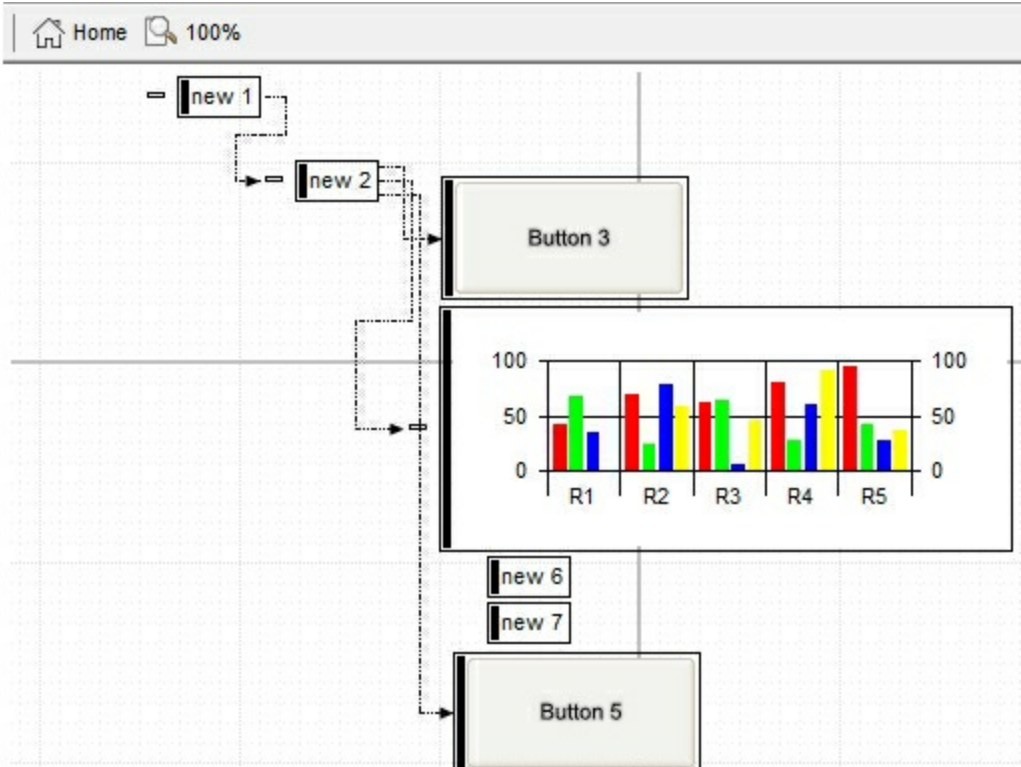
The control supports ActiveX hosting, or in other words, any element can host another inside controls. The License property indicates a string expression that indicates the runtime license key for the component being inserted, if required. Only, the vendor of the component you are going to use is able to give you such of runtime license, so please contact the control's vendor for such of key. Your development license key is not compatible with the runtime license key, so it can't be used here. The [Control](#) property has effect only if the element's [Type](#) property is set on exElementHostControl. If you insert a runtime-licensed control you must specify the License property before calling the Control property. Use the [ElementFormat](#) property to specify the area where the inner control is displayed. The [Object](#) property returns a reference to newly created control. The ExSurface control fires the [OLEEvent](#) event if an inside ActiveX control fires an event.

The Control property must be formatted in one of the following ways:

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

**The look and feel of the inner ActiveX control depends on the identifier you are using, and the version of the library that implements the ActiveX control, so you need to consult the documentation of the inner ActiveX control you are inserting inside the exSurface control.**

The following screen shot shows the surface with different inner controls:



# property Element.MaxHeight as Long

Specifies the maximum height of the element.

Type	Description
Long	A Long expression that specifies the max height of the element. If negative, the MaxHeight property has no effect.

By default, the MaxHeight property is -1, which indicates that it has no effect. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element. The [Height](#) property specifies the height of the element. The [Width](#) property specifies the width of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size.

# property Element.MaxWidth as Long

Specifies the maximum width of the element.

Type	Description
Long	A Long expression that specifies the max width of the element. If negative, the MaxWidth property has no effect.

By default, the MaxWidth property is -1, which indicates that it has no effect. The [MinWidth](#)/MaxWidth and [MinHeight](#)/[MaxHeight](#) properties specifies the min/max size of the element. The [Height](#) property specifies the height of the element. The [Width](#) property specifies the width of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size.

# property Element.MinHeight as Long

Specifies the minimum height of the element.

Type	Description
Long	A Long expression that specifies the min height of the element.

By default, the MinHeight property is 10 pixels. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element. The [Height](#) property specifies the height of the element. The [Width](#) property specifies the width of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size.

# property Element.MinWidth as Long

Specifies the minimum width of the element.

Type	Description
Long	A Long expression that specifies the min width of the element, in pixles.

By default, the MinWidth property is 10 pixels. The MinWidth/[MaxWidth](#) and [MinHeight](#)/[MaxHeight](#) properties specifies the min/max size of the element. The [Height](#) property specifies the height of the element. The [Width](#) property specifies the width of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size.



# method Element.MoveTo (X as Long, Y as Long, Width as Long, Height as Long)

Moves the element to a new position.

Type	Description
X as Long	A long expression that specifies the x-position of the element on the surface.
Y as Long	A long expression that specifies the y-position of the element on the surface.
Width as Long	A long expression that specifies the width of the element on the surface.
Height as Long	A long expression that specifies the height of the element on the surface.

Use the MoveTo method to move/resize the element to a new position / size. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The [X](#) and [Y](#) properties specifies the position of the element on the surface. The [Width](#) property specifies the width of the element. The [Height](#) property specifies the height of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element.

# property Element.NextSiblingChild as Element

Retrieves the next sibling of the element in the parent's child list

Type	Description
<a href="#">Element</a>	Indicates the next sibling element.

Use the [FirstChild](#) and [NextSiblingChild](#) properties to enumerate child elements one by one. The [FirstChild](#) property returns nothing if the current element contains no child elements. The [FirstChild](#) property returns the first child element. The [LastChild](#) property indicates the last child element. Use the [LastChild](#) and [PrevSiblingChild](#) properties to backward enumerate all child elements. The [NextVisibleChild](#) property indicates the next visible element. The [PrevVisibleChild](#) property indicates the previously visible element. The [Children](#) property retrieves the child elements at once. The [ChildCount](#) property specifies the number of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the [Element](#) object specifies whether the element supports adding child elements at runtime.

# property Element.NextVisibleChild as Element

Retrieves the next visible element in the parent's child list

Type	Description
<a href="#">Element</a>	Indicates the next visible element.

The NextVisibleChild property indicates the next visible element. The [PrevVisibleChild](#) property indicates the previously visible element. The [FirstChild](#) property returns nothing if the current element contains no child elements. The FirstChild property returns the first child element. Use the FirstChild and [NextSiblingChild](#) properties to enumerate child elements one by one. The [LastChild](#) property indicates the last child element. Use the [LastChild](#) and [PrevSiblingChild](#) properties to backward enumerate all child elements. The [Children](#) property retrieves the child elements at once. The [ChildCount](#) property specifies the number of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime.

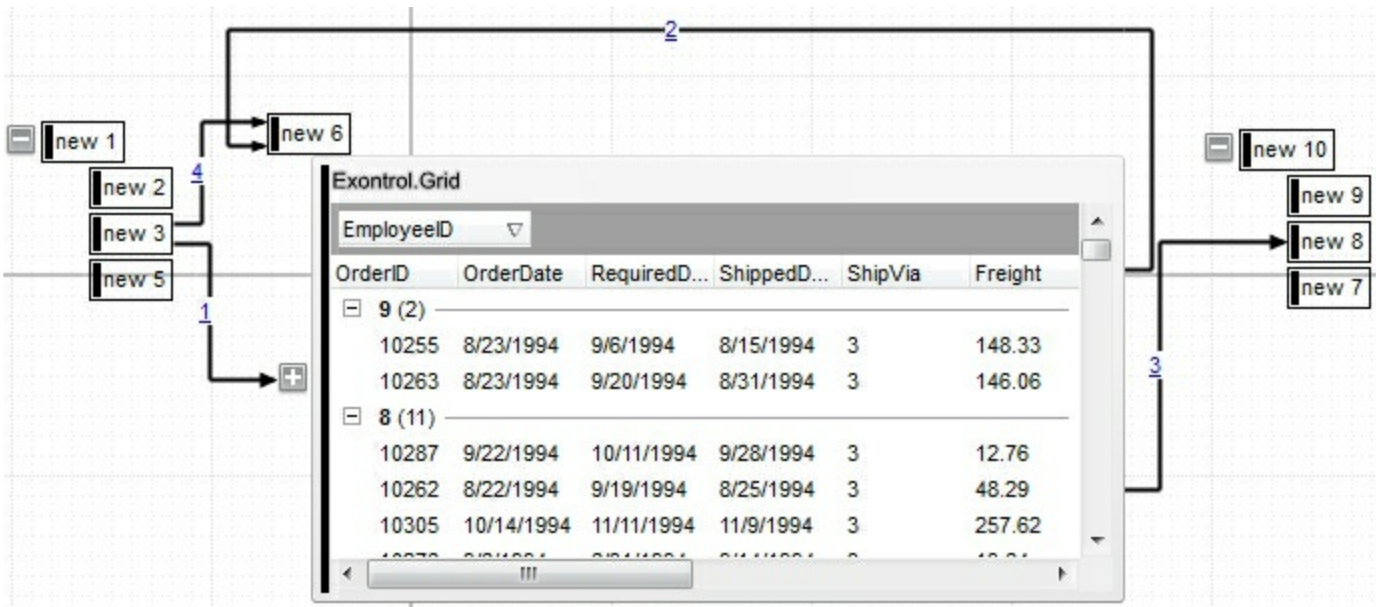
# property Element.Object as Object

Returns the inner object hosted by the current element.

Type	Description
Object	A reference to inner ActiveX control.

The Object property returns nothing if element hosts no ActiveX or creating it failed. The Object property returns a reference to newly created control. The ExSurface control fires the [OLEEvent](#) event if an inside ActiveX control fires an event. Use the [ElementFormat](#) property to specify the area where the inner control is displayed. The [InsertControl](#) property adds a new Element object with the [Type](#) set on exElementHostControl, and so it hosts an ActiveX inside. The [Control](#) property returns the control's identifier. The [License](#) property specifies the runtime-license of the control. For instance, if the Control property is "Exontrol.Button", the Object property returns a reference to an Exontrol.Button object. If the Control property is "Exontrol.Grid" the Object property returns a reference to an Exontrol.Grid object.

The following screen shot shows the Exontrol.Grid on the surface:



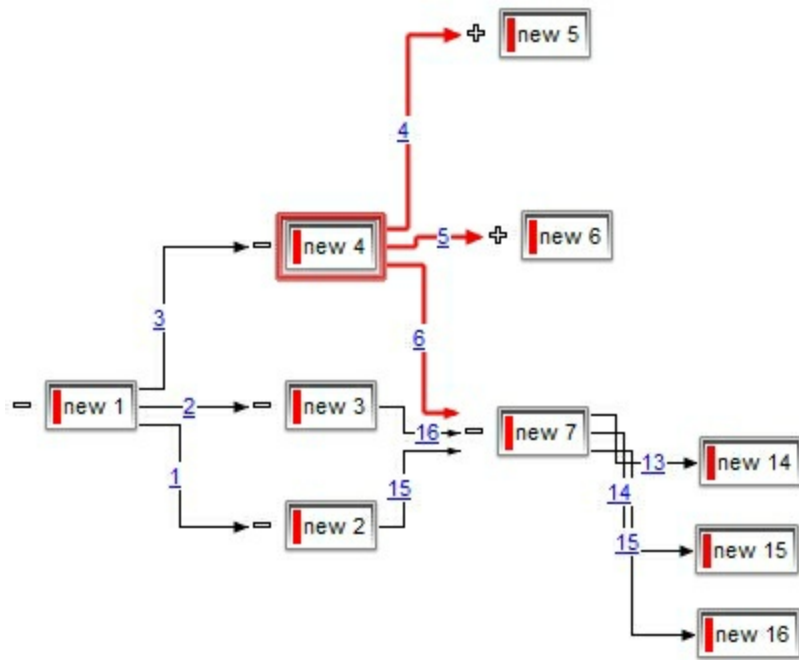
# property Element.OutgoingLinks as Variant

Returns a safe array of outgoing links.

Type	Description
Variant	A safe-array of <a href="#">Link</a> objects that specifies the links that starts from the current element. You can use the for-each statement to enumerate all outgoing links.

The OutgoingLinks property specifies the list of links that starts from the current element. The [IncomingLinks](#) property specifies the list of links that ends on the current element. The [ElementFrom](#) property of the Link object indicates where the Link starts. The [ElementTo](#) property of the Link object indicates where the Link ends. Use the [ShowLinksColor](#)(exShowLinksStartsFrom)/[ShowLinksStyle](#)(exShowLinksStartsFrom)/[ShowLinksWidth](#)(exShowLinksStartsFrom) properties to mark the outgoing links of selected elements. The [PathTo](#) property indicates if a path exists from current element to specified element.

The following screen shot shows the outgoing links (red ):



The following VB sample enumerates the outgoing elements ( of selected elements ):

```
Private Sub Surface1_SelectionChanged()  
    With Surface1  
        Dim s As Variant  
        For Each s In .Selection  
            Debug.Print "Outgoing Elements of " & s.ID & "are: "  
            With s
```

```
For Each i In .OutgoingLinks
```

```
    Debug.Print i.ElementTo.ID
```

```
Next
```

```
End With
```

```
Next
```

```
End With
```

```
End Sub
```

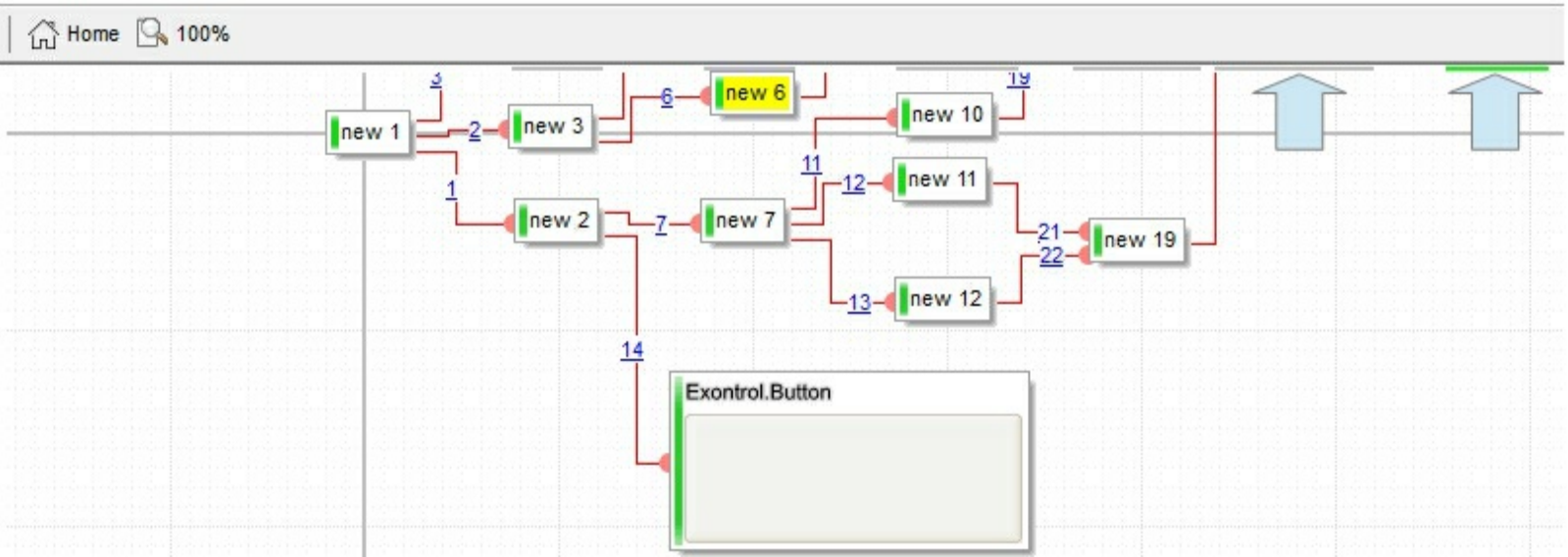
# property Element.OverviewColor as Color

Gets or sets a value that indicates the element's overview color.

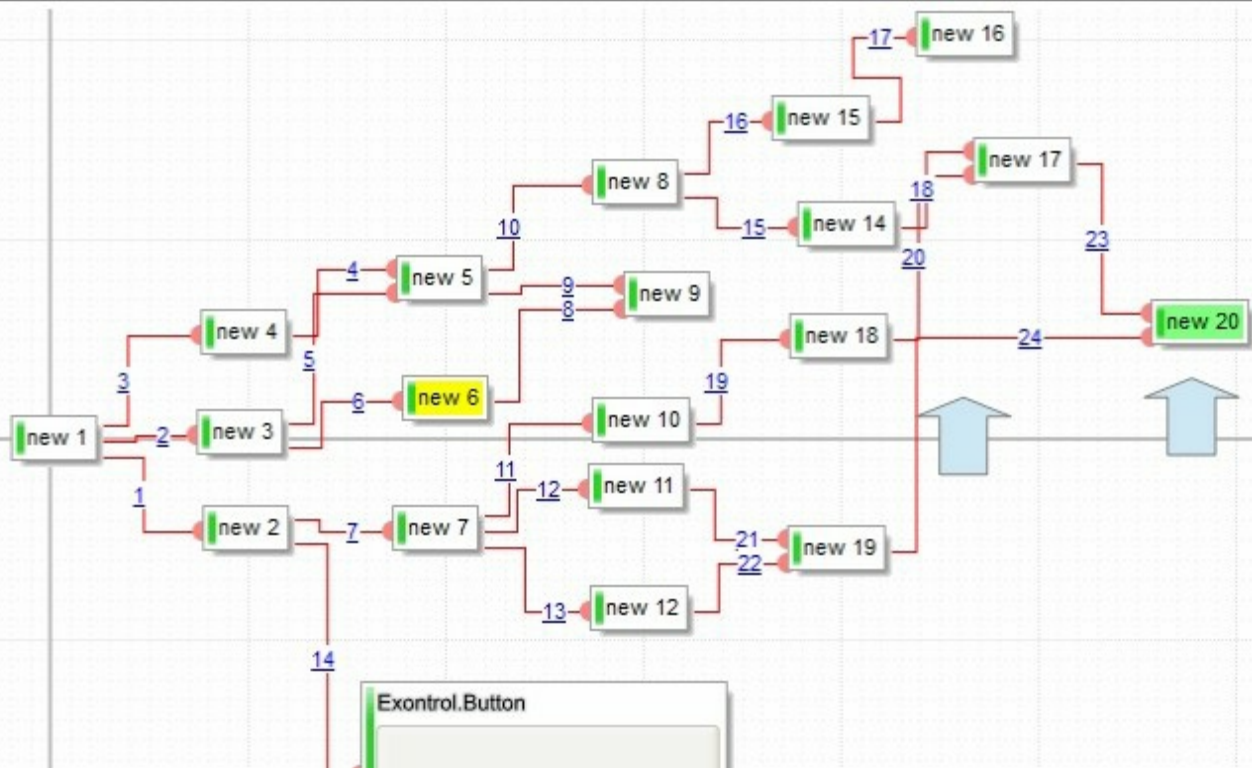
Type	Description
Color	A Color expression that specifies the color to show the element when it is not visible on the surface's client area

By default, the OverviewColor property is -1, which indicates that the control's [OverviewColor](#) property specifies the overview-color to show the elements on the control's border. Use the OverviewColor property to specify a different color to be shown for specific elements. The OverviewColor property has effect when the element is not fitting the surface's client area and it is shown on the border of the surface. If the control's [OverviewColor](#) property is -1, no elements is shown on the border.

The following screen shot shows the how elements are shown when they are not visible in the surface's client area ( look on the border ) :



The following screen shot shows the elements when they are visible on the surface's client area:





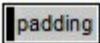
# property Element.Padding(Edge as PaddingEdgeEnum) as Long

Returns or sets a value that indicates the padding of the element's background.

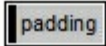
Type	Description
Edge as <a href="#">PaddingEdgeEnum</a>	A PaddingEdgeEnum expression that specifies the edge to be changed
Long	A long expression that defines the padding

By default, Padding property is 0. The Padding property specifies the body/background padding. The [BorderPadding](#) property specifies the padding to be applied on borders, to define the position of the status and the body parts of the element. The [StatusPadding](#) property specifies the status padding. The [ClientPadding](#) property specifies the padding while the element's [Type](#) property is exElementHostWindow or exElementHostControl.

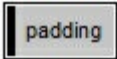
The following screen shot shows the element when Padding is 0 (default):



The following screen shot shows the element when Padding is 1:



The following screen shot shows the element when Padding is 4:



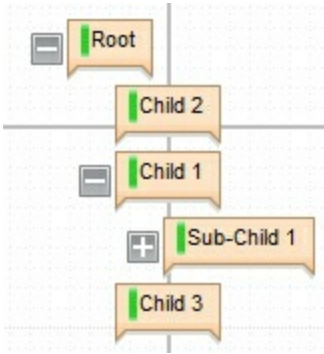
# property Element.Parent as Element

Specifies the element's parent.

Type	Description
<a href="#">Element</a>	An Element object that specifies the parent element.

The Parent property indicates the element's parent. By default, the Parent property is nothing, which indicates that the element has no parent. Use the Parent property to change the element's parent. The [ParentChangeEvent](#) event occurs when the element's parent is changed. Use the [AllowInsertObject](#) property to specify whether the user can change the element's parent at runtime by dragging the element over the other. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime. The [Children](#) property specifies the list of child elements. The [Level](#) property specifies the level in the hierarchy of the element.

The following screen shot shows the elements arranged as a tree:



# property Element.PathTo (ElementTo as Element) as Boolean

Determines if there is any path from the current element to the specified element.

Type	Description
ElementTo as <a href="#">Element</a>	An Element object that specifies the ending element
Boolean	A Boolean expression that specifies if a path exists between current element to specified element. A Path is defined by the element and its outgoing links.

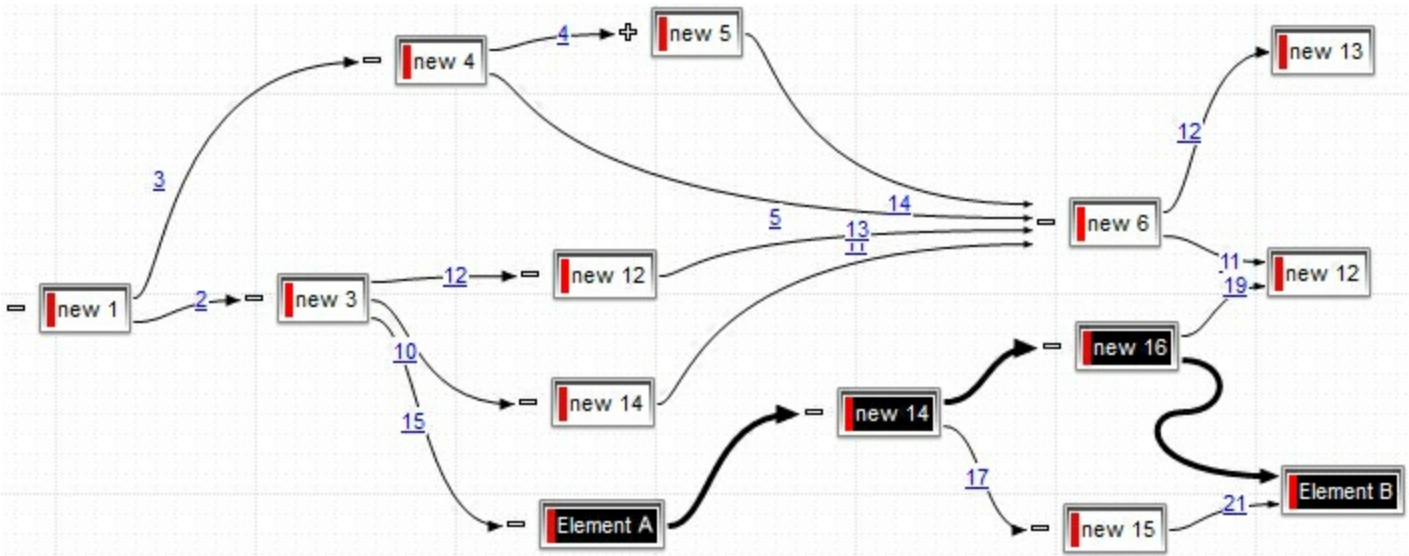
The PathTo property indicates if a path exists from current element to specified element. The [OutgoingLinks](#) property specifies the list of links that starts from the current element. The [IncomingLinks](#) property specifies the list of links that ends on the current element. The [ElementFrom](#) property of the Link object indicates where the Link starts. The [ElementTo](#) property of the Link object indicates where the Link ends. Use the [ShowLinksColor](#)(exShowLinksStartsFrom)/[ShowLinksStyle](#)(exShowLinksStartsFrom)/[ShowLinksWidth](#)(exShowLinksStartsFrom) properties to mark the outgoing links of selected elements.

For instance, you can use the PathTo property to prevent adding cycles in the chart, as in the following VB sample:

```
Private Sub Surface1_AllowLink(ByVal ElementFrom As EXSURFACELibCtl.IElement, ByVal ElementTo As EXSURFACELibCtl.IElement, Cancel As Boolean)
    Cancel = ElementTo.PathTo(ElementFrom)
End Sub
```

The [AllowLink](#) event notifies that the user links two elements.

The following screen shot shows the path between Element A and Element B:





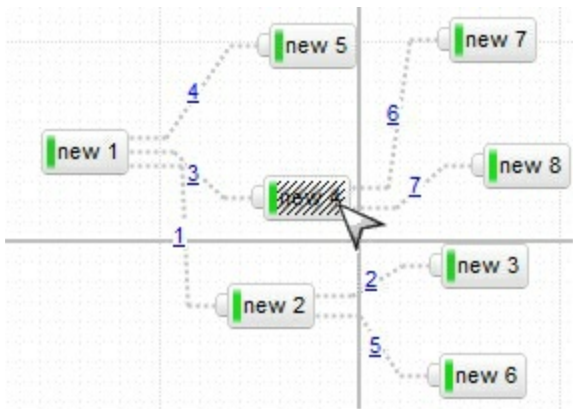
# property Element.Pattern as Pattern

Specifies the pattern to be shown on the element's background.

Type	Description
Pattern	A Pattern to be shown over the element's background.

By default, no Pattern is shown on the element's background. The Pattern property defines the pattern to be shown on the control's background. The [BackColor](#) property specifies the element's background color. [Background](#)(exElementBackColor) property specifies the default background color / visual appearance. The [ForeColor](#) property specifies the element's foreground color. The [Picture](#) property specifies the picture to be shown on the element's background. The [PictureDisplay](#) property specifies the way the element's picture is displayed on the element's background.

The following screen shot shows an element with different pattern:



The following samples applies a pattern like ( not available ) over the element:

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

```
With Surface1
  With .Elements
    With .Add("Element+Pattern",-100).Pattern
      .Type = 6
      .Color = RGB(224,224,224)
    End With
    .Add "Element",100
  End With
End With
```

```

With Surface1
  With .Elements
    With .Add("Element+ Pattern",-100).Pattern
      .Type = exPatternBDiagonal
      .Color = RGB(224,224,224)
    End With
    .Add "Element",100
  End With
End With

```

## VB.NET

```

With Exsurface1
  With .Elements
    With .Add("Element+ Pattern",-100).Pattern
      .Type = excontrol.EXSURFACELib.PatternEnum.exPatternBDiagonal
      .Color = Color.FromArgb(224,224,224)
    End With
    .Add("Element",100)
  End With
End With

```

## VB.NET for /COM

```

With AxSurface1
  With .Elements
    With .Add("Element+ Pattern",-100).Pattern
      .Type = EXSURFACELib.PatternEnum.exPatternBDiagonal
      .Color = RGB(224,224,224)
    End With
    .Add("Element",100)
  End With
End With

```

## C++

```

/*
Copy and paste the following directives to your header file as

```

it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control Library'

```
#import <ExSurface.dll>
using namespace EXSURFACELib;
*/
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
> GetControlUnknown();
EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();
    EXSURFACELib::IPatternPtr var_Pattern = var_Elements-
> Add("Element+Pattern",long(-100),vtMissing,vtMissing,vtMissing,vtMissing)-
> GetPattern();
    var_Pattern->PutType(EXSURFACELib::exPatternBDiagonal);
    var_Pattern->PutColor(RGB(224,224,224));
    var_Elements->Add("Element",long(100),vtMissing,vtMissing,vtMissing,vtMissing);
```

## C++ Builder

```
Exsurfacelib_tlb::IElementsPtr var_Elements = Surface1->Elements;
    Exsurfacelib_tlb::IPatternPtr var_Pattern = var_Elements-
> Add(TVariant("Element+Pattern"),TVariant(-100),TNoParam(),TNoParam(),TNoParam())
> Pattern;
    var_Pattern->Type = Exsurfacelib_tlb::PatternEnum::exPatternBDiagonal;
    var_Pattern->Color = RGB(224,224,224);
    var_Elements-
> Add(TVariant("Element"),TVariant(100),TNoParam(),TNoParam(),TNoParam(),TNoParam());
```

## C#

```
exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;
    exontrol.EXSURFACELib.Pattern var_Pattern =
var_Elements.Add("Element+Pattern",-100,null,null,null,null).Pattern;
    var_Pattern.Type = exontrol.EXSURFACELib.PatternEnum.exPatternBDiagonal;
    var_Pattern.Color = Color.FromArgb(224,224,224);
    var_Elements.Add("Element",100,null,null,null,null);
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
    var var_Elements = Surface1.Elements;
    var var_Pattern =
var_Elements.Add("Element+ Pattern",-100,null,null,null,null).Pattern;
    var_Pattern.Type = 6;
    var_Pattern.Color = 14737632;
    var_Elements.Add("Element",100,null,null,null,null);
</SCRIPT>
```

## C# for /COM

```
EXSURFACELib.Elements var_Elements = axSurface1.Elements;
    EXSURFACELib.Pattern var_Pattern =
var_Elements.Add("Element+ Pattern",-100,null,null,null,null).Pattern;
    var_Pattern.Type = EXSURFACELib.PatternEnum.exPatternBDiagonal;
    var_Pattern.Color =
(uint)ColorTranslator.ToWin32(Color.FromArgb(224,224,224));
    var_Elements.Add("Element",100,null,null,null,null);
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Element,com_Elements,com_Pattern;
    anytype var_Element,var_Elements,var_Pattern;
    ;

    super();

    var_Elements = exsurface1.Elements(); com_Elements = var_Elements;
```



```

    var_Element =
COM::createFromObject(com_Elements.Add("Element+Pattern",COMVariant::createFrom
com_Element = var_Element;
    var_Pattern = com_Element.Pattern(); com_Pattern = var_Pattern;
    com_Pattern.Type(6/*exPatternBDiagonal*/);
    com_Pattern.Color(WinApi::RGB2int(224,224,224));
    com_Elements.Add("Element",COMVariant::createFromInt(100));
}

```

## Delphi 8 (.NET only)

```

with AxSurface1 do
begin
    with Elements do
    begin
        with Add('Element+Pattern',TObject(-100),Nil,Nil,Nil,Nil).Pattern do
        begin
            Type := EXSURFACELib.PatternEnum.exPatternBDiagonal;
            Color := $e0e0e0;
        end;
        Add('Element',TObject(100),Nil,Nil,Nil,Nil);
    end;
end

```

## Delphi (standard)

```

with Surface1 do
begin
    with Elements do
    begin
        with Add('Element+Pattern',OleVariant(-100),Null,Null,Null,Null).Pattern do
        begin
            Type := EXSURFACELib_TLB.exPatternBDiagonal;
            Color := $e0e0e0;
        end;
        Add('Element',OleVariant(100),Null,Null,Null,Null);
    end;
end

```

## VFP

```
with thisform.Surface1
  with .Elements
    with .Add("Element+Pattern",-100).Pattern
      .Type = 6
      .Color = RGB(224,224,224)
    endwhile
    .Add("Element",100)
  endwhile
endwith
```

## dBASE Plus

```
local oSurface,var_Elements,var_Pattern

oSurface = form.Activex1.nativeObject
var_Elements = oSurface.Elements
var_Pattern = var_Elements.Add("Element+Pattern",-100).Pattern
var_Pattern.Type = 6
var_Pattern.Color = 0xe0e0e0
var_Elements.Add("Element",100)
```

## XBasic (Alpha Five)

```
Dim oSurface as P
Dim var_Elements as P
Dim var_Pattern as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
var_Elements = oSurface.Elements
var_Pattern = var_Elements.Add("Element+Pattern",-100).Pattern
var_Pattern.Type = 6
var_Pattern.Color = 14737632
var_Elements.Add("Element",100)
```

## Visual Objects

local var\_Elements as IElements

local var\_Pattern as IPattern

var\_Elements := oDCOCX\_Exontrol1:Elements

var\_Pattern := var\_Elements:Add("Element+Pattern",-100,nil,nil,nil,nil):**Pattern**

var\_Pattern.Type := exPatternBDiagonal

var\_Pattern.Color := RGB(224,224,224)

var\_Elements:Add("Element",100,nil,nil,nil,nil)

## PowerBuilder

OleObject oSurface,var\_Elements,var\_Pattern

oSurface = ole\_1.Object

var\_Elements = oSurface.Elements

var\_Pattern = var\_Elements.Add("Element+Pattern",-100).**Pattern**

var\_Pattern.Type = 6

var\_Pattern.Color = RGB(224,224,224)

var\_Elements.Add("Element",100)

## Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voElements

Get ComElements to voElements

Handle hoElements

Get Create (RefClass(cComElements)) to hoElements

Set pvComObject of hoElements to voElements

Variant voElement

Get ComAdd of hoElements "Element+Pattern" -100 Nothing Nothing Nothing

Nothing to voElement

Handle hoElement

Get Create (RefClass(cComElement)) to hoElement

Set pvComObject of hoElement to voElement

```

Variant voPattern
Get ComPattern of hoElement to voPattern
Handle hoPattern
Get Create (RefClass(cComPattern)) to hoPattern
Set pvComObject of hoPattern to voPattern
    Set ComType of hoPattern to OLExPatternBDiagonal
    Set ComColor of hoPattern to (RGB(224,224,224))
Send Destroy to hoPattern
Send Destroy to hoElement
Get ComAdd of hoElements "Element" 100 Nothing Nothing Nothing Nothing
to Nothing
Send Destroy to hoElements
End_Procedure

```

## XBase++

```

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

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oElements
    LOCAL oPattern
    LOCAL oSurface

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oSurface := XbpActiveXControl():new( oForm:drawingArea )
    oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
    oSurface:create(,, {10,60},{610,370} )

    oElements := oSurface:Elements()

```

```
oPattern := oElements:Add("Element+Pattern",-100):Pattern()
oPattern.Type := 6/*exPatternBDiagonal*/
oPattern.SetProperty("Color",AutomationTranslateColor( GraMakeRGBColor
( { 224,224,224 } ) , .F. ))
oElements:Add("Element",100)

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp.handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

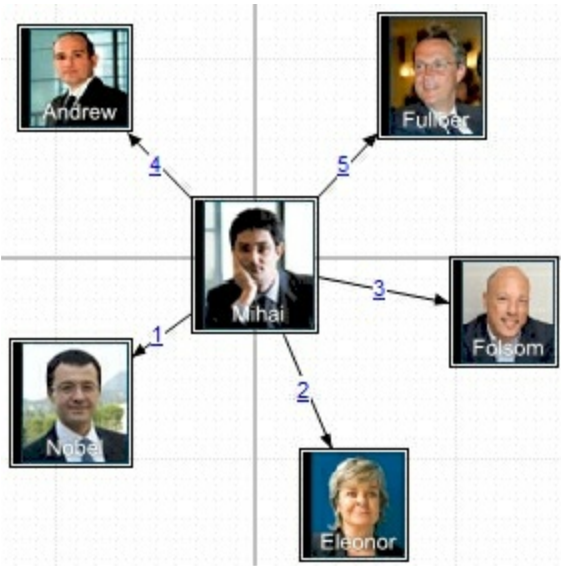
# property Element.Picture as IPictureDisp

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

Type	Description
IPictureDisp	A Picture object to be shown on the element's background.

The Picture property specifies the picture to be shown on the element's background. The [PictureDisplay](#) property specifies the way the element's picture is displayed on the element's background. The [BackColor](#) property specifies the element's background color. [Background](#)(exElementBackColor) property specifies the default background color / visual appearance. The [ForeColor](#) property specifies the element's foreground color. The [Pattern](#) property defines the pattern to be shown on the control's background. The [Padding](#) property defines the padding of the element. The [BorderColor](#) property specifies the color to show the border for a specific element. The [StatusColor](#) property specifies the color or the visual appearance to show the element's status part. Use the [Pictures](#) / [ExtraPictures](#) properties to display different pictures on the element. Use the [Picture](#) property to assign your logo on the control's background.

The following screen shot shows the element's background filled with the Picture property:



# property Element.PictureDisplay as PictureDisplayEnum

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

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

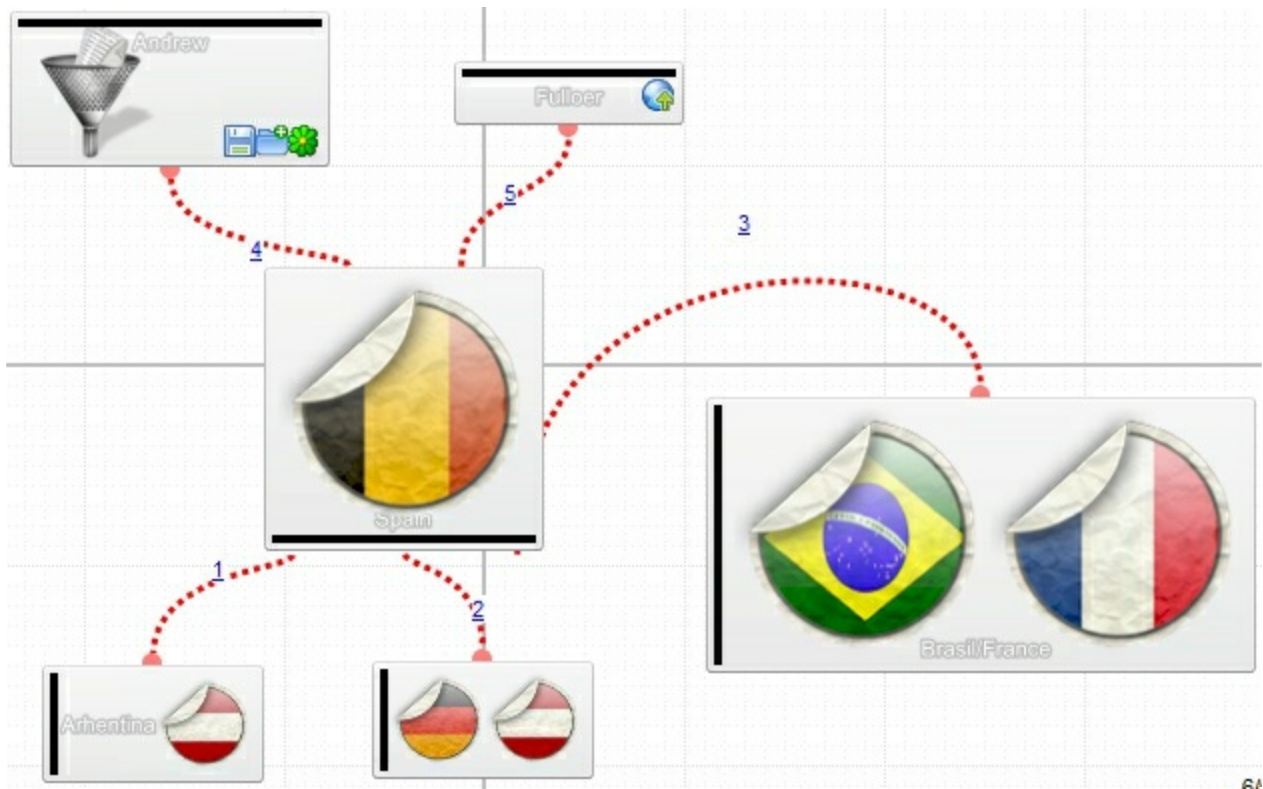
The PictureDisplay property specifies the way the element's picture is displayed on the element's background. The [Picture](#) property specifies the picture to be shown on the element's background. The [BackColor](#) property specifies the element's background color. [Background](#)(exElementBackColor) property specifies the default background color / visual appearance. The [ForeColor](#) property specifies the element's foreground color. The [Pattern](#) property defines the pattern to be shown on the control's background. The [Padding](#) property defines the padding of the element. The [BorderColor](#) property specifies the color to show the border for a specific element. The [StatusColor](#) property specifies the color or the visual appearance to show the element's status part.

# property Element.Pictures as String

Specifies the list of pictures to be displayed on the element.

Type	Description
String	A string expression that specifies the list of pictures to be shown on the element's body. The element's body can display one ore more pictures at the time, on different lines. For instance: "1,2/pic1" displays the 1 and 2 icons on the first line, while pic2 is displayed on the second line.

By default, the Pictures property is "", which means that initially no pictures are being displayed on the element. The Pictures or/and [ExtraPictures](#) property displays a collection of icons, pictures in the element's body. The Picture and ExtraPictures may display one or more pictures at the time. The , character indicates the separator of pictures in the same line, while the / character divides the lines to show the pictures. For instance, "1,2" displays icon with the index 1 and 2 on the same line, while the "1/2,pic1" displays the first icon on the first line, the second icon and the picture pic1 on the second line. The [Images](#) method loads icons to the control. The Images collection can display only 16x16 icons. The [HTMLPicture](#) assigns a key to a picture object. The [ShowHandCursorOn](#) property specifies whether the hand cursor is shown when hovering a picture on the element. The [HandCursorClick](#) event occurs once the user clicks a picture on the element ( [ShowHandCursorOn](#) property must include the exShowHandCursorPicture, exShowHandCursorIcon and exShowHandCursorPictures ). The [PicturesAlign](#) property specifies the alignment of the Pictures relative to the element.





The [Picture](#) property displays a picture on the element's background. The [PictureDisplay](#) property specifies the way the element's picture is displayed on the element's background. The [BackColor](#) property specifies the element's background color. [Background](#)(exElementBackColor) property specifies the default background color / visual appearance. The [ForeColor](#) property specifies the element's foreground color. The [Pattern](#) property defines the pattern to be shown on the control's background.

# property Element.PicturesAlign as ContentAlignmentEnum

Indicates the alignment of the element's picture.

Type	Description
<a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that specifies the alignment of the pictures in the element.

By default, The PicturesAlign property is exBottomRight. The PicturesAlign property specifies the alignment of the Pictures relative to the element. By default, the [Pictures](#) property is "", which means that initially no pictures are being displayed on the element. The [Pictures](#) or/and [ExtraPictures](#) property displays a collection of icons, pictures in the element's body. The Picture and ExtraPictures may display one or more pictures at the time. The , character indicates the separator of pictures in the same line, while the / character divides the lines to show the pictures. For instance, "1,2" displays icon with the index 1 and 2 on the same line, while the "1/2,pic1" displays the first icon on the first line, the second icon and the picture pic1 on the second line. The [Images](#) method loads icons to the control. The Images collection can display only 16x16 icons. The [HTMLPicture](#) assigns a key to a picture object. The [ShowHandCursorOn](#) property specifies whether the hand cursor is shown when hovering a picture on the element. The [HandCursorClick](#) event occurs once the user clicks a picture on the element ( [ShowHandCursorOn](#) property must include the exShowHandCursorPicture, exShowHandCursorIcon and exShowHandCursorPictures ).

# property Element.PrevSiblingChild as Element

Retrieves the prev sibling of the element in the parent's child list

Type	Description
<a href="#">Element</a>	Indicates the previously sibling element.

Use the [LastChild](#) and PrevSiblingChild properties to backward enumerate all child elements. Use the [FirstChild](#) and NextSiblingChild properties to enumerate child elements one by one. The [FirstChild](#) property returns nothing if the current element contains no child elements. The [FirstChild](#) property returns the first child element. The [LastChild](#) property indicates the last child element. The [NextVisibleChild](#) property indicates the next visible element. The [PrevVisibleChild](#) property indicates the previously visible element. The [Children](#) property retrieves the child elements at once. The [ChildCount](#) property specifies the number of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime.

# property Element.PrevVisibleChild as Element

Retrieves the prev visible element in the parent's child list

Type	Description
<a href="#">Element</a>	Indicates the previously visible element.

The PrevVisibleChild property indicates the previously visible element. The [NextVisibleChild](#) property indicates the next visible element. The [FirstChild](#) property returns nothing if the current element contains no child elements. The FirstChild property returns the first child element. Use the FirstChild and [NextSiblingChild](#) properties to enumerate child elements one by one. The [LastChild](#) property indicates the last child element. Use the [LastChild](#) and [PrevSiblingChild](#) properties to backward enumerate all child elements. The [Children](#) property retrieves the child elements at once. The [ChildCount](#) property specifies the number of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime.

# property Element.Resizable as Boolean

Gets or sets a value that indicates whether the user can resize the element.

Type	Description
Boolean	A Boolean expression that specifies whether the user can resize the element at runtime.

By default, the Resizable property is True, which specifies that the user can resize the element at runtime. The Resizable property has no effect if the element's [AutoSize](#) property is True. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element. The [Height](#) property specifies the height of the element. The [Width](#) property specifies the width of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size. The [Selectable](#) property indicates whether the user can select the element. Use the [AllowResizeObject](#) property to specify whether the element can be resized at runtime.

# method Element.ScrollTo (To as ContentAlignmentEnum)

Moves or scrolls the surface, so the current element aligns to the specified corner.

Type	Description
To as <a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that specifies where on the surface the element should be scrolled to.

The ScrollTo method ensures that the element fits the surface's visible area. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position. Use the [ScrollTo](#) method of the control to scroll the surface at specified position. The [MovePoint](#) method of the control moves the surface from the one point to another. The [MoveCorner](#) method scrolls the surface from a corner to another.

The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size. The [Width](#) property specifies the width of the element. The [Height](#) property specifies the height of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element.

# property Element.Selectable as Boolean

Indicates if the element is selectable.

Type	Description
Boolean	A Boolean expression that specifies whether the element is selectable or un-selectable.

By default, the Selectable property is True. The Selectable property of the Element object indicates whether the element is selectable or un-selectable. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. Use the [Enabled](#) property to show the element as disabled ( grayed ).

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [AllowMoveObject](#) property specifies the keys combination so the user can move the element from the cursor. The [AllowMoveSelection](#) property indicates whether the entire selection is moved if an element in the selection is moved. Set the AllowMoveObject property on exDisallow, to prevent user to move any element in the surface.

# property Element.Selected as Boolean

Indicates if the element is selected or unselected.

Type	Description
Boolean	A Boolean expression that specifies whether the element is selected or unselected.

By default, the Selected property is False. The Selected property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background\(exSelectObjectRectColor\)](#) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.



# method Element.SendToBack ()

Sends the element to the back.

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

The SendToBack method sends the current element to the back. The [BringToFront](#) method brings the element to the front. For instance, if two element gets intersected, you can use the [BringToFront](#) method to bring one element on front, or SendToBack method to send the element on the back. The [BringToFront](#) and SendToBack methods changes the drawing order of the elements.

# property Element.ShowCheckBox as Boolean

Gets or sets a value that indicates whether the element shows or hides the check-box.

Type	Description
Boolean	A Boolean expression that specifies whether the element displays the checkbox.

By default, the ShowCheckBox property is False, which means that no check-box is displayed. Use the ShowCheckBox property to show or hide the element's checkbox. Use the [CheckBoxAlign](#) property to align the element's checkbox. Use the [Checked](#) property to specify the state of the element's checkbox. The [CheckElement](#) event occurs when the checkbox's state is changed. Use the [Background](#)(exCheckBoxState0), [Background](#)(exCheckBoxState1), [Background](#)(exCheckBoxState2) to change the visual appearance for all check-boxes.

# property Element.ShowHandCursorOn as ShowHandCursorOnEnum

Specifies whether the hand cursor is shown when hovering the element.

Type	Description
<a href="#">ShowHandCursorOnEnum</a>	A ShowHandCursorOnEnum expression that specifies the parts of the element where the hand cursor is shown when the mouse-pointer hovers it.

By default, the ShowHandCursorOn property is exShowHandCursorAnchorAll, which indicates that the hand cursor is shown when user hovers any anchor element (<a>). Use the [Caption](#) or [ExtraCaption](#) property to display hyperlinks or anchors in the element. The [AnchorClick](#) event is fired once the user clicks an anchor element. The control fires the [HandCursorClick](#) event when the user clicks a part of the element. The Hit parameter of the [HandCursorClick](#) specifies the part of the element being clicked, while the Key parameter specifies a value associated with the part being clicked as listed bellow:

- exShowHandCursorCheck -> key specifies the [Element.Checked](#) property.
- exShowHandCursorAnchor -> key specifies the identifier of the anchor element such as <a id;options> anchor </a>
- exShowHandCursorPicture -> key specifies the name of the picture being clicked ( [HTMLPicture](#) property )
- exShowHandCursorIcon - key specifies the index of the icon being clicked ( [Images](#) method )

The above flags can be combined with the following flags:

- exShowHandCursorCaption, indicates that the part being clicked belong to the element's caption.
- exShowHandCursorExtraCaption, indicates that the part being clicked belong to the element's extra caption.
- exShowHandCursorPictures, indicates that the part being clicked belong to the element's pictures.
- exShowHandCursorExtraPictures, indicates that the part being clicked belong to the element's extra pictures.

The following samples shows how you can handle clicking an icon or a picture of the element:

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

' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.

Private Sub Surface1\_HandCursorClick(ByVal Element As Object,ByVal Hit As

```
Long,ByVal Key As Variant)
```

```
With Surface1
```

```
    Debug.Print( Key )
```

```
End With
```

```
End Sub
```

```
With Surface1
```

```
    .Images
```

```
"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr  
& _
```

```
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
& _
```

```
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
& _
```

```
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vI8j4f/qfEZeB  
& _
```

```
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="
```

```
    .HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
```

```
    .HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
```

```
With .Elements.Add("Caption")
```

```
    .Pictures = "1,2/pic1/pic2"
```

```
    .PicturesAlign = 33
```

```
    .ShowHandCursorOn = 771 '
```

```
ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or
```

```
ShowHandCursorOnEnum.exShowHandCursorPictures Or
```

```
ShowHandCursorOnEnum.exShowHandCursorIcon Or
```

```
ShowHandCursorOnEnum.exShowHandCursorPicture
```

```
    .CaptionAlign = 1
```

```
End With
```

```
End With
```

## VB6

**' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
Private Sub Surface1_HandCursorClick(ByVal Element As
```

```
EXSURFACELibCtl.IElement,ByVal Hit As
```

```
EXSURFACELibCtl.ShowHandCursorOnEnum,ByVal Key As Variant)
```

```
With Surface1
```

```
    Debug.Print( Key )
```

```
End With
```

```
End Sub
```

```
With Surface1
```

```
    .Images
```

```
"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr  
& _
```

```
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
& _
```

```
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
& _
```

```
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
& _
```

```
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="
```

```
    .HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
```

```
    .HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
```

```
With .Elements.Add("Caption")
```

```
    .Pictures = "1,2/pic1/pic2"
```

```
    .PicturesAlign = exBottomCenter
```

```
    .ShowHandCursorOn =
```

```
ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or
```

```
ShowHandCursorOnEnum.exShowHandCursorPictures Or
```

```
ShowHandCursorOnEnum.exShowHandCursorIcon Or
```

```
ShowHandCursorOnEnum.exShowHandCursorPicture
```

```
    .CaptionAlign = exTopCenter
```

```
End With
```

```
End With
```

## VB.NET

' **HandCursorClick event** - The uses clicks a part of the element that shows the had cursor.

```
Private Sub Exsurface1_HandCursorClick(ByVal sender As System.Object,ByVal  
Element As exontrol.EXSURFACELib.Element,ByVal Hit As
```

exontrol.EXSURFACELib.ShowHandCursorOnEnum,ByVal Key As Object) Handles

Exsurface1.**HandCursorClick**

With Exsurface1

Debug.Print( Key )

End With

End Sub

With Exsurface1

.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oI

& \_

"/oFBoVDolFo1HpFJpVLplNp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl

& \_

"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m

& \_

"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB

& \_

"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=")

.set\_HTMLPicture("pic1","c:\exontrol\images\zipdisk.gif")

.set\_HTMLPicture("pic2","c:\exontrol\images\auction.gif")

With .Elements.Add("Caption")

.Pictures = "1,2/pic1/pic2"

.PicturesAlign =

exontrol.EXSURFACELib.ContentAlignmentEnum.exBottomCenter

**.ShowHandCursorOn =**

exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures

Or exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures Or

exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon Or

exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture

.CaptionAlign = exontrol.EXSURFACELib.ContentAlignmentEnum.exTopCenter

End With

End With

## VB.NET for /COM

**' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

Private Sub AxSurface1\_HandCursorClick(ByVal sender As System.Object, ByVal e As AxEXSURFACELib.\_ISurfaceEvents\_HandCursorClickEvent) Handles

AxSurface1.**HandCursorClick**

With AxSurface1

Debug.Print( e.key )

End With

End Sub

With AxSurface1

.Images("gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oI  
& \_

"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
& \_

"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
& \_

"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
& \_

"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=")

.set\_HTMLPicture("pic1","c:\exontrol\images\zipdisk.gif")

.set\_HTMLPicture("pic2","c:\exontrol\images\auction.gif")

With .Elements.Add("Caption")

.Pictures = "1,2/pic1/pic2"

.PicturesAlign = EXSURFACELib.ContentAlignmentEnum.exBottomCenter

**.ShowHandCursorOn =**

EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or

EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures Or

EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon Or

EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture

.CaptionAlign = EXSURFACELib.ContentAlignmentEnum.exTopCenter

End With

End With

**C++**

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

void OnHandCursorClickSurface1(LPDISPATCH Element,long Hit,VARIANT Key)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control
Library'
        #import <ExSurface.dll>
        using namespace EXSURFACELib;
    */
    EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
>GetControlUnknown();
    OutputDebugStringW( L"Key" );
}

EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
>GetControlUnknown();
spSurface1-
> Images(_bstr_t("gBJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIA
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="));
spSurface1->PutHTMLPicture(L"pic1","c:\\exontrol\\images\\zipdisk.gif");
spSurface1->PutHTMLPicture(L"pic2","c:\\exontrol\\images\\auction.gif");
EXSURFACELib::IElementPtr var_Element = spSurface1->GetElements()-
>Add("Caption",vtMissing,vtMissing,vtMissing,vtMissing,vtMissing);
var_Element->PutPictures(L"1,2/pic1/pic2");
var_Element->PutPicturesAlign(EXSURFACELib::exBottomCenter);
var_Element-
> PutShowHandCursorOn(EXSURFACELib::ShowHandCursorOnEnum(EXSURFACELib::e
| EXSURFACELib::exShowHandCursorPictures |
EXSURFACELib::exShowHandCursorIcon | EXSURFACELib::exShowHandCursorPicture));
var_Element->PutCaptionAlign(EXSURFACELib::exTopCenter);

```



## C++ Builder

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
void __fastcall TForm1::Surface1HandCursorClick(TObject
*Sender,Exsurfacelib_tlb::IElement
*Element,Exsurfacelib_tlb::ShowHandCursorOnEnum Hit,Variant Key)
{
    OutputDebugString( L"Key" );
}

Surface1-
> Images(TVariant(String("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIE
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA="));
Surface1->HTMLPicture[L"pic1"] = TVariant("c:\\exontrol\\images\\zipdisk.gif");
Surface1->HTMLPicture[L"pic2"] = TVariant("c:\\exontrol\\images\\auction.gif");
Exsurfacelib_tlb::IElementPtr var_Element = Surface1->Elements-
>Add(TVariant("Caption"),TNoParam(),TNoParam(),TNoParam(),TNoParam(),TNoParam(

    var_Element->Pictures = L"1,2/pic1/pic2";
    var_Element->PicturesAlign =
Exsurfacelib_tlb::ContentAlignmentEnum::exBottomCenter;
    var_Element->ShowHandCursorOn =
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorExtraPictures |
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorPictures |
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorIcon |
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorPicture;
    var_Element->CaptionAlign =
```

Exsurface1lib\_tlb::ContentAlignmentEnum::exTopCenter;

C#

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
private void exsurface1_HandCursorClick(object
sender,exontrol.EXSURFACELib.Element
Element,exontrol.EXSURFACELib.ShowHandCursorOnEnum Hit,object Key)
{
    System.Diagnostics.Debug.Print( Key.ToString() );
}
```

**//this.exsurface1.HandCursorClick += new  
exontrol.EXSURFACELib.exg2antt.HandCursorClickEventHandler(this.exsurface1**

```
exsurface1.Images("gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaA  
+  
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
+  
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
+  
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
+  
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");  
exsurface1.set_HTMLPicture("pic1","c:\\exontrol\\images\\zipdisk.gif");  
exsurface1.set_HTMLPicture("pic2","c:\\exontrol\\images\\auction.gif");  
exontrol.EXSURFACELib.Element var_Element =  
exsurface1.Elements.Add("Caption",null,null,null,null,null);  
var_Element.Pictures = "1,2/pic1/pic2";  
var_Element.PicturesAlign =  
exontrol.EXSURFACELib.ContentAlignmentEnum.exBottomCenter;  
var_Element.ShowHandCursorOn =  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures |  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures |  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon |
```

```
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture;  
    var_Element.CaptionAlign =  
exontrol.EXSURFACELib.ContentAlignmentEnum.exTopCenter;
```

## JavaScript

```
<SCRIPT FOR="Surface1" EVENT="HandCursorClick(Element,Hit,Key)"  
LANGUAGE="JScript">  
    alert( Key );  
</SCRIPT>  
  
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"  
id="Surface1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
  
Surface1.Images("gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAJEZFEaIEaEEaAI/  
+  
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
+  
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
+  
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
+  
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=");  
Surface1.HTMLPicture("pic1") = "c:\\exontrol\\images\\zipdisk.gif";  
Surface1.HTMLPicture("pic2") = "c:\\exontrol\\images\\auction.gif";  
var var_Element = Surface1.Elements.Add("Caption",null,null,null,null,null);  
    var_Element.Pictures = "1,2/pic1/pic2";  
    var_Element.PicturesAlign = 33;  
    var_Element.ShowHandCursorOn = 771;  
    var_Element.CaptionAlign = 1;  
</SCRIPT>
```

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
private void axSurface1_HandCursorClick(object sender,
AxEXSURFACELib._ISurfaceEvents_HandCursorClickEvent e)
{
    System.Diagnostics.Debug.Print( e.key.ToString() );
}
```

**//this.axSurface1.HandCursorClick += new  
AxEXSURFACELib.\_ISurfaceEvents\_HandCursorClickEventHandler(this.axSurface1**

```
axSurface1.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEa/
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");
axSurface1.set_HTMLPicture("pic1","c:\\exontrol\\images\\zipdisk.gif");
axSurface1.set_HTMLPicture("pic2","c:\\exontrol\\images\\auction.gif");
EXSURFACELib.Element var_Element =
axSurface1.Elements.Add("Caption",null,null,null,null,null);
    var_Element.Pictures = "1,2/pic1/pic2";
    var_Element.PicturesAlign =
EXSURFACELib.ContentAlignmentEnum.exBottomCenter;
    var_Element.ShowHandCursorOn =
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures |
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures |
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon |
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture;
    var_Element.CaptionAlign = EXSURFACELib.ContentAlignmentEnum.exTopCenter;
```

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
void onEvent_HandCursorClick(COM _Element,int _Hit,COMVariant _Key)
{
    ;
    print( _Key );
}

public void init()
{
    COM com_Element;
    anytype var_Element;
    str var_s;
    ;

    super();

    var_s =
    "gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

    var_s = var_s +
    "oFBoVDolFo1HpFJpVLplNp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu'

    var_s = var_s +
    "wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0

    var_s = var_s +
    "3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeBC

    var_s = var_s +
    "AOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=";
    exsurface1.Images(COMVariant::createFromStr(var_s));
    exsurface1.HTMLPicture("pic1","c:\\exontrol\\images\\zipdisk.gif");
    exsurface1.HTMLPicture("pic2","c:\\exontrol\\images\\auction.gif");
    var_Element = COM::createFromObject(exsurface1.Elements()).Add("Caption");
```

```

com_Element = var_Element;
com_Element.Pictures("1,2/pic1/pic2");
com_Element.PicturesAlign(33/*exBottomCenter*/);
com_Element.ShowHandCursorOn(771/*exShowHandCursorExtraPictures |
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture*/);
com_Element.CaptionAlign(1/*exTopCenter*/);
}

```

## Delphi 8 (.NET only)

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

procedure TForm1.AxSurface1_HandCursorClick(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_HandCursorClickEvent);
begin
  with AxSurface1 do
  begin
    OutputDebugString( e.key );
  end
end;

with AxSurface1 do
begin

Images('gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oLL
+
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu\
+
'wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0
+
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0
+
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=');
set_HTMLPicture('pic1','c:\exontrol\images\zipdisk.gif');

```

```

set_HTMLPicture('pic2','c:\exontrol\images\auction.gif');
with Elements.Add('Caption',Nil,Nil,Nil,Nil,Nil) do
begin
    Pictures := '1,2/pic1/pic2';
    PicturesAlign := EXSURFACELib.ContentAlignmentEnum.exBottomCenter;
    ShowHandCursorOn :=
Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures)
Or Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures) Or
Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon) Or
Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture);
    CaptionAlign := EXSURFACELib.ContentAlignmentEnum.exTopCenter;
end;
end

```

## Delphi (standard)

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

procedure TForm1.Surface1HandCursorClick(ASender: TObject; Element : IElement;Hit
: ShowHandCursorOnEnum;Key : OleVariant);
begin
    with Surface1 do
    begin
        OutputDebugString( Key );
    end
end;

with Surface1 do
begin

Images('gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIL
+
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu\
+
'wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0

```

```

+
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0
+
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=');
HTMLPicture['pic1'] := 'c:\exontrol\images\zipdisk.gif';
HTMLPicture['pic2'] := 'c:\exontrol\images\auction.gif';
with Elements.Add('Caption',Null,Null,Null,Null,Null) do
begin
    Pictures := '1,2/pic1/pic2';
    PicturesAlign := EXSURFACELib_TLB.exBottomCenter;
    ShowHandCursorOn :=
Integer(EXSURFACELib_TLB.exShowHandCursorExtraPictures) Or
Integer(EXSURFACELib_TLB.exShowHandCursorPictures) Or
Integer(EXSURFACELib_TLB.exShowHandCursorIcon) Or
Integer(EXSURFACELib_TLB.exShowHandCursorPicture);
    CaptionAlign := EXSURFACELib_TLB.exTopCenter;
end;
end

```

## VFP

```

*** HandCursorClick event - The uses clicks a part of the element that shows the had
cursor. ***
LPARAMETERS Element,Hit,Key
    with thisform.Surface1
        DEBUGOUT( Key )
    endwith

with thisform.Surface1
    var_s =
'gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

    var_s = var_s +
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu'

    var_s = var_s +

```



"wGBwWDwmFw2HxGJxWLxmNx0xiFdyOTh8Tf9ZymXx+ QytcyNgz8r0ObIWjyWds+mC

```
var_s = var_s +
```

"3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeBC

```
var_s = var_s +
```

"AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA="

```
.Images(var_s)
```

```
.Object.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
```

```
.Object.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
```

```
with .Elements.Add("Caption")
```

```
.Pictures = "1,2/pic1/pic2"
```

```
.PicturesAlign = 33
```

```
.ShowHandCursorOn = 771 &&
```

ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or

ShowHandCursorOnEnum.exShowHandCursorPictures Or

ShowHandCursorOnEnum.exShowHandCursorIcon Or

ShowHandCursorOnEnum.exShowHandCursorPicture

```
.CaptionAlign = 1
```

```
endwith
```

```
endwith
```

## dBASE Plus

```
/*  
with (this.ACTIVEX1.nativeObject)  
    HandCursorClick = class::nativeObject_HandCursorClick  
endwith  
*/
```

**// The uses clicks a part of the element that shows the had cursor.**

```
function nativeObject_HandCursorClick(Element,Hit,Key)
```

```
    local oSurface
```

```
    oSurface = form.Activex1.nativeObject
```

```
    ? Str(Key)
```

```
return
```

```
local oSurface,var_Element
```

```

oSurface = form.ActiveX1.NativeObject
oSurface.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAI/

oSurface.Template = [HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"] //
oSurface.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
oSurface.Template = [HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"] //
oSurface.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
var_Element = oSurface.Elements.Add("Caption")
    var_Element.Pictures = "1,2/pic1/pic2"
    var_Element.PicturesAlign = 33
    var_Element.ShowHandCursorOn = 771 /*exShowHandCursorExtraPictures |
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture*/
    var_Element.CaptionAlign = 1

```

## XBasic (Alpha Five)

' **The uses clicks a part of the element that shows the had cursor.**

```

function HandCursorClick as v (Element as OLE::Exontrol.Surface.1::IElement, Hit as
OLE::Exontrol.Surface.1::ShowHandCursorOnEnum, Key as A)

```

```

    Dim oSurface as P
    oSurface = topparent:CONTROL_ACTIVEX1.activex
    ? Key
end function

```

```

Dim oSurface as P
Dim var_Element as P

```

```

oSurface = topparent:CONTROL_ACTIVEX1.activex
oSurface.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAI/

oSurface.Template = "HTMLPicture(\"pic1\") = \"c:\exontrol\images\zipdisk.gif\""" '
oSurface.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
oSurface.Template = "HTMLPicture(\"pic2\") = \"c:\exontrol\images\auction.gif\""" '
oSurface.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
var_Element = oSurface.Elements.Add("Caption")

```

```

var_Element.Pictures = "1,2/pic1/pic2"
var_Element.PicturesAlign = 33
var_Element.ShowHandCursorOn = 771 'exShowHandCursorExtraPictures +
exShowHandCursorPictures + exShowHandCursorIcon +
exShowHandCursorPicture
var_Element.CaptionAlign = 1

```

## Visual Objects

```

METHOD OCX_Exontrol1HandCursorClick(Element,Hit,Key) CLASS MainDialog
    // HandCursorClick event - The uses clicks a part of the element that shows
the had cursor.
    OutputDebugString(String2Psz( AsString(Key) ))
RETURN NIL

local var_Element as IElement

oDCOCX_Exontrol1:Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFE

oDCOCX_Exontrol1:[HTMLPicture,"pic1"] := "c:\exontrol\images\zipdisk.gif"
oDCOCX_Exontrol1:[HTMLPicture,"pic2"] := "c:\exontrol\images\auction.gif"
var_Element := oDCOCX_Exontrol1:Elements:Add("Caption",nil,nil,nil,nil,nil)
var_Element.Pictures := "1,2/pic1/pic2"
var_Element.PicturesAlign := exBottomCenter
var_Element.ShowHandCursorOn := exShowHandCursorExtraPictures |
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture
var_Element.CaptionAlign := exTopCenter

```

## PowerBuilder

```

/*begin event HandCursorClick(oleobject Element,long Hit,any Key) - The uses clicks a
part of the element that shows the had cursor.*/
/*
OleObject oSurface
oSurface = ole_1.Object
MessageBox("Information",string( String(Key) ))

```

```

*/
/*end event HandCursorClick*/

OleObject oSurface,var_Element

oSurface = ole_1.Object
oSurface.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

oSurface.HTMLPicture("pic1","c:\exontrol\images\zipdisk.gif")
oSurface.HTMLPicture("pic2","c:\exontrol\images\auction.gif")
var_Element = oSurface.Elements.Add("Caption")
    var_Element.Pictures = "1,2/pic1/pic2"
    var_Element.PicturesAlign = 33
    var_Element.ShowHandCursorOn = 771 /*exShowHandCursorExtraPictures |
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture*/
    var_Element.CaptionAlign = 1

```

## Visual DataFlex

```

// The uses clicks a part of the element that shows the had cursor.
Procedure OnComHandCursorClick Variant IIElement OLEShowHandCursorOnEnum
IIEHit Variant IKey
    Forward Send OnComHandCursorClick IIElement IIEHit IKey
    ShowIn IKey
End_Procedure

Procedure OnCreate
    Forward Send OnCreate
    Send ComImages
    "gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

    Set ComHTMLPicture "pic1" to "c:\exontrol\images\zipdisk.gif"
    Set ComHTMLPicture "pic2" to "c:\exontrol\images\auction.gif"
    Variant voElements
    Get ComElements to voElements
    Handle hoElements

```

```

Get Create (RefClass(cComElements)) to hoElements
Set pvComObject of hoElements to voElements
Variant voElement
Get ComAdd of hoElements "Caption" Nothing Nothing Nothing Nothing
Nothing to voElement
Handle hoElement
Get Create (RefClass(cComElement)) to hoElement
Set pvComObject of hoElement to voElement
Set ComPictures of hoElement to "1,2/pic1/pic2"
Set ComPicturesAlign of hoElement to OLEexBottomCenter
Set ComShowHandCursorOn of hoElement to
(OLEexShowHandCursorExtraPictures + OLEexShowHandCursorPictures +
OLEexShowHandCursorIcon + OLEexShowHandCursorPicture)
Set ComCaptionAlign of hoElement to OLEexTopCenter
Send Destroy to hoElement
Send Destroy to hoElements
End_Procedure

```

## XBase++

```

PROCEDURE OnHandCursorClick(oSurface,Element,Hit,Key)
  DevOut( Transform(Key,"") )
RETURN

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

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oElement
  LOCAL oSurface

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( „{100,100}, {640,480}„, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

```

```

oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(,, {10,60},{610,370} )

oSurface:HandCursorClick := {|Element,Hit,Key|
OnHandCursorClick(oSurface,Element,Hit,Key)} /*The uses clicks a part of the element
that shows the had cursor.*/

oSurface:Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEalEaEEaAlA/

oSurface:SetProperty("HTMLPicture","pic1","c:\exontrol\images\zipdisk.gif")
oSurface:SetProperty("HTMLPicture","pic2","c:\exontrol\images\auction.gif")
oElement := oSurface:Elements():Add("Caption")
oElement:Pictures := "1,2/pic1/pic2"
oElement:PicturesAlign := 33/*exBottomCenter*/
oElement:ShowHandCursorOn :=
771/*exShowHandCursorExtraPictures+exShowHandCursorPictures+exShowHandCurs

oElement:CaptionAlign := 1/*exTopCenter*/

oForm:Show()
DO WHILE nEvent != xbeP_Quit
nEvent := AppEvent( @mp1, @mp2, @oXbp )
oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

# property Element.StartUpdateElement as Long

Starts changing properties of the element, so EndUpdateElement method adds programmatically updated properties to undo/redo queue.

Type	Description
Long	A Long expression that specifies the handle to be passed to <a href="#">EndUpdateElement</a> so the updated properties of the bar are added to the Undo/Redo queue of the chart, so they can be used in undo/redo operations.

The StartUpdateElement/[EndUpdateElement](#) methods record and add changes of the current element to the control's Undo/Redo queue. You can use the [StartBlockUndoRedo](#) / [EndBlockUndoRedo](#) methods to group multiple Undo/Redo operations into a single-block. The [AllowUndoRedo](#) property specifies whether the control supports undo/redo operations for objects (elements, links, ...). No entry is added to the Undo/Redo queue if no property is changed for the current element. Each call of the StartUpdateElement must be succeeded by a [EndUpdateElement](#) call. The [UndoListAction](#) property lists the Undo actions that can be performed in the chart. The [RedoListAction](#) property lists the Redo actions that can be performed in the chart.

The StartUpdateElement/[EndUpdateElement](#) methods can record changes for the following properties only:

- [Type](#), defines the element's type (default and control)
- [Control](#), Specifies the identifier of the inner control hosted by the current element (only if the Type is exElementHostControl)
- [License](#), Indicates the runtime license required to create the inner control (only if the Type is exElementHostControl)
- [Caption](#), gets or sets a value that indicates the HTML caption to be displayed on the element
- [Parent](#), defines the element's parent (when the element is part of a hierarchy)
- [ChildPosition](#), specifies the position of the element while it is a child element (when the element is part of a hierarchy)
- [ID](#), defines the element's unique identifier
- [Format](#), specifies the way the control shows the parts of the element
- [AutoSize](#), specifies if the element computes its size automatically
- [ShowCheckBox](#), shows or hides the element's check-box
- [X](#), specifies the element's x-position
- [Y](#), specifies the element's y-position
- [Width](#), specifies the width of the element
- [Height](#), specifies the height of the element
- [Expanded](#), expands or collapses an element (by default, the UI expand/collapse of the

elements are not recorded into the Undo/Redo queue)

- [Visible](#), shows or hides the element
- [Enabled](#), enables or disables the element
- [Checked](#), checks or un-checks the element (by default, the UI check/uncheck of the elements are not recorded into the Undo/Redo queue)
- [BackColor](#), gets or sets a value that indicates the element's background color
- [ForeColor](#), gets or sets a value that indicates the element's foreground color
- [OverviewColor](#), gets or sets a value that indicates the element's overview color
- [BorderColor](#), gets or sets a value that indicates the element's border color
- [StatusColor](#), gets or sets a value that indicates the element's status color
- [Pictures](#), specifies the list of pictures to be displayed on the element
- [ExtraPictures](#), specifies the list of additional pictures to be displayed on the element
- [ExtraCaption](#), gets or sets a value that indicates additional HTML caption to be displayed on the element
- [Pattern](#), specifies the pattern to be shown on the element's background
- [UserData](#), associates any extra data associated with the element

The Undo/Redo records show as:

- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the StartUpdateElement / [EndUpdateElement](#) methods

within the [UndoListAction/RedoListAction](#) result.



# property Element.StatusAlign as EdgeAlignmentEnum

Specifies the alignment of the status inside the element.

Type	Description
<a href="#">EdgeAlignmentEnum</a>	An EdgeAlignmentEnum expression that specifies the alignment of the element's status

The StatusAlign property indicates the alignment of the element's status. Use the [StatusColor](#) property to specify a different status color. The [StatusPadding](#) property specifies the status padding. The [BackColor](#) property specifies the element's background color. The [ForeColor](#) property specifies the element's foreground color. The [StatusSize](#) property indicates the size of the element's status. The [StatusPattern](#) property indicates a different pattern to be displayed on the element's status part.

# property Element.StatusColor as Color

Gets or sets a value that indicates the element's status color.

Type	Description
Color	A Color expression that specifies the color to show the element's status. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used to paint the part. Use the <a href="#">Add</a> method to add new skins to the control.

By default, the StatusColor is -1, which indicates that the default status color is applied. The [Background](#)(exElementStatusColor) property specifies the default status color / visual appearance. Use the StatusColor property to specify a different status color. The [StatusPadding](#) property specifies the status padding. The [BackColor](#) property specifies the element's background color. The [ForeColor](#) property specifies the element's foreground color. The [StatusSize](#) property indicates the size of the element's status. The [StatusAlign](#) property indicates the alignment of the element's status. The [StatusPattern](#) property indicates a different pattern to be displayed on the element's status part.

# property Element.StatusPadding(Edge as PaddingEdgeEnum) as Long

Returns or sets a value that indicates the padding of the element's status.

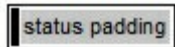
Type	Description
Edge as <a href="#">PaddingEdgeEnum</a>	A PaddingEdgeEnum expression that specifies the edge to be changed
Long	A long expression that defines the padding

By default, the StatusPadding is 0. The StatusPadding property specifies the status padding. Use the [StatusColor](#) property to specify a different status color. The [BackColor](#) property specifies the element's background color. The [ForeColor](#) property specifies the element's foreground color. The [StatusSize](#) property indicates the size of the element's status. The [StatusAlign](#) property indicates the alignment of the element's status. The [StatusPattern](#) property indicates a different pattern to be displayed on the element's status part.

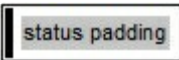
The following screen shot shows the element when StatusPadding is 0 (default):



The following screen shot shows the element when StatusPadding is 1:



The following screen shot shows the element when StatusPadding is 4:



# property Element.StatusPattern as Pattern

Specifies the pattern of the element's status

Type	Description
Pattern	A <a href="#">Pattern</a> object that defines the pattern to be shown on the status

The StatusPattern property indicates a different pattern to be displayed on the element's status part. Use the [StatusColor](#) property to specify a different status color. The [StatusPadding](#) property specifies the status padding. The [BackColor](#) property specifies the element's background color. The [ForeColor](#) property specifies the element's foreground color. The [StatusSize](#) property indicates the size of the element's status. The [StatusAlign](#) property indicates the alignment of the element's status.

# property Element.StatusSize as Long

Specifies the size of the status inside the element.

Type	Description
Long	A Long expression that defines the size of the status part of the element.

By default, the StatusSize property is 4. The StatusSize property indicates the size of the element's status. The [StatusAlign](#) property indicates the alignment of the element's status. Use the [StatusColor](#) property to specify a different status color. The [StatusPadding](#) property specifies the status padding. The [BackColor](#) property specifies the element's background color. The [ForeColor](#) property specifies the element's foreground color. The [StatusPattern](#) property indicates a different pattern to be displayed on the element's status part.

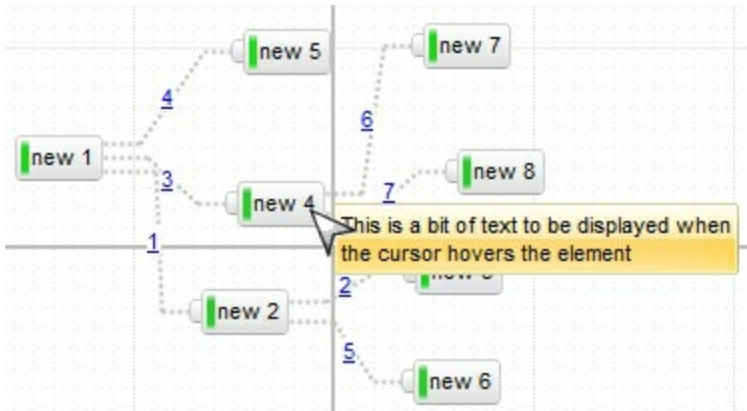
# property Element.ToolTip as String

Gets or sets a value (tooltip) that's displayed once the cursor hovers the element.

Type	Description
String	A String expression that defines the element's tooltip.

By default, the ToolTip property is empty. The ToolTip of the Element is shown, if the ToolTip property is not empty and the cursor hovers the element. Use the [ToolTipTitle](#) property to assign a title for the element's tooltip. Use the [ShowToolTip](#) method to programmatically show a custom tooltip. 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.

The following screen shot shows the element's tooltip when the cursor hovers the element:



The ToolTip property supports the following HTML elements:

- `<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.

The control supports expandable HTML captions feature which allows you to

expand(show)/collapse(hide) different information using `<a ;exp=>` or `<a ;e64=>` anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY</a>`" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY" string encodes the "`<fgcolor 808080>show lines<a>-</a></fgcolor>`" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "`<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3`" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **<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=rrgbb> ... </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=rrgbb> ... </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=rrgbb> ... </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=rrgbb> ... </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 Element.ToolTipTitle as String

Gets or sets a value (title) that's displayed once the cursor hovers the element.

Type	Description
String	A String expression that defines the title of the element's tooltip.

By default, the ToolTipTitle property is empty. Use the ToolTipTitle property to assign a title for the element's tooltip. The [ToolTip](#) of the Element is shown, if the ToolTip property is not empty and the cursor hovers the element. Use the [ShowToolTip](#) method to programmatically show a custom tooltip. 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 Element.Type as ElementHostTypeEnum

Specifies the element's type.

Type	Description
<a href="#">ElementHostTypeEnum</a>	An ElementHostTypeEnum expression that defines type of the element.

By default, the Type property is exElementHostDefault, which defines the default node, with caption, check, pictures and so on. Use the Type property to change the element's type. Changing the Type property at runtime, loses the properties like Caption, Pictures, and so on. So setting the Type property must be done before calling or changing any of the other properties. Use the [Add/Insert](#) method to add/insert an (child) element of exElementHostDefault type. Use the [InsertControl](#) method to add a new element whose Type property is set on exElementHostControl. Use the [Caption/ExtraCaption](#) property to specify the element's caption or extra-caption. Use the [Pictures/ExtraPictures](#) property to display one or more icons, picture to the element. Use the [ShowCheckBox](#) property to show the element's check box. Use the [ElementFormat](#) property to define how the parts of the element are displayed.

In case, the Type property is:

- **exElementHostWindow**, you must use the [Window](#) property to associate an existent window with the current element.
- **exElementHostControl**, you must use the [Control/License](#) property to create an inner control or an ActiveX control. Use the [ElementFormat](#) property to define how the parts of the element are displayed. The control fires the [OLEEvent](#) event if an inside ActiveX control fires an event.

# property Element.userData as Variant

Indicates any extra data associated with the element.

Type	Description
Variant	A VARIANT expression that defines the extra-data associated with the element.

By default, the element's UserData property is empty, so no extra-data is associated with the node. You can use the UserData property to associate your extra data with the current element. The UserData property is not used by the control in any other way, it is provided to store your extra data.

# property Element.Visible as Boolean

Shows or hides the element.

Type	Description
Boolean	A Boolean expression that specifies whether the element is visible or hidden on the surface.

By default, the Visible property is True. Use the Visible property to hide an element. Use the [Remove](#) method to remove an element. If the element is child of another element ( [Parent](#) property specifies the element's parent ), the element is visible, if all parents are visible and expanded. The [Expanded](#) property specifies whether an element is expanded or collapsed. If the element is not visible, any incoming or outgoing links are not visible as well. The [OutgoingLinks](#) property specifies the list of links that starts from the current element. The [IncomingLinks](#) property specifies the list of links that ends on the current element.

# property Element.VisibleChildCount as Long

Counts the number of visible child elements.

Type	Description
Long	A Long expression that specifies the number of visible child elements.

The VisibleChildCount property specifies the list of visible children. The [ChildCount](#) property counts the number of child elements. The [VisibleChildren](#) property specifies the list of visible child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime. The [ParentChangeEvent](#) event occurs when the element's parent is changed.

# property Element.VisibleChildren as Variant

Returns a safe array of visible child elements.

Type	Description
Variant	A Safe-Array of elements indicating the list of child elements. You can use the for-each statement to enumerate the child elements of specified node.

The Visible Children property specifies the list of visible child elements. The [Visible](#) property specifies whether the element is visible or hidden. The [Children](#) property specifies the list of child elements. Use the [Parent](#) property to change the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime. The [ParentChangeEvent](#) event occurs when the element's parent is changed.

# property Element.Width as Long

Specifies the width of the element.

Type	Description
Long	A Long expression that specifies the width in pixels of the element.

The Width property specifies the width of the element. The [Height](#) property specifies the height of the element. Use the Width and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the Width and the [Height](#) can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element. The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size.



# property Element.Window as Long

Returns or sets the handle of the window to be hosted by the element.

Type	Description
Long	A HANDLE expression that specifies the handle of the window to be hosted by the current element.

By default, the Windows property is 0. The Window property has effect only, if the element's [Type](#) property is set on **exElementHostWindow**. The Window property may be used to associate an existent window from your form or dialog to be hosted by an element on the surface. Use the [InsertControl](#) method to add a new element whose Type property is set on exElementHostControl, and so to host a new ActiveX control.

# property Element.X as Long

Specifies the element's x-position.

Type	Description
Long	A Long expression that specifies the x-position of the element on the surface.

The X and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size. The [Width](#) property specifies the width of the element. The [Height](#) property specifies the height of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element.

# property Element.Y as Long

Specifies the element's y-position.

Type	Description
Long	A Long expression that specifies the y-position of the element on the surface.

The [X](#) and [Y](#) properties specifies the position of the element on the surface. Use the [MoveTo](#) method to move/resize the element to a new position / size. The [Width](#) property specifies the width of the element. The [Height](#) property specifies the height of the element. Use the [Width](#) and [Height](#) properties to resize the element, while the [AutoSize](#) property is False. If [AutoSize](#) property is True, the [Width](#) and the [Height](#) can not be changed programmatically. The [MinWidth/MaxWidth](#) and [MinHeight/MaxHeight](#) properties specifies the min/max size of the element.

# Elements object

The Elements collection holds the elements to be shown on the surface. The [Elements](#) property of the control accesses the Elements collection. The [Links](#) property of the control accesses the [Links](#) collection. The Elements collection supports the following properties and methods:

Name	Description
<a href="#">Add</a>	Adds an Element 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>	Returns the number of elements in the collection.
<a href="#">Insert</a>	Inserts a child Element object to the collection and returns a reference to the newly created object.
<a href="#">InsertControl</a>	Inserts a child Element object ( that hosts another control inside ) to the collection and returns a reference to the newly created object.
<a href="#">Item</a>	Returns a specific Element of the Elements collection, giving its identifier.
<a href="#">Remove</a>	Removes a specific member from the Elements collection, giving its identifier or reference.

# method Elements.Add ([Caption as Variant], [X as Variant], [Y as Variant])

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

Type	Description
Caption as Variant	A String expression that specifies the caption to be displayed on the element. The Caption property specifies the value of the <a href="#">Caption</a> parameter. Use the CaptionAlign property to align the element's caption.
X as Variant	A long expression that specifies the x-position where the element is sown. If missing, the element is shown at (0,0) position on the surface. The <a href="#">X</a> property indicates the element's x-position on the surface.
Y as Variant	A long expression that specifies the y-position where the element is sown. If missing, the element is shown at (0,0) position on the surface. The <a href="#">X</a> property indicates the element's x-position on the surface.
Return	Description
<a href="#">Element</a>	An Element object that represents the newly created and added element.

The Add method adds programmatically a new element to the surface. Use the [Insert](#) method to insert programmatically a child element. Use the [InsertControl](#) method to insert programmatically a child element that hosts an inner ActiveX control. The control fires the [AddElement](#) event once a new element is added to the Elements collection. The [AutoSize](#) property specifies whether the element's size if computed automatically based on its content. While the AutoSize property is True, the element is not resizable. Use the [Width](#) / [Height](#) property to specify the size of the element. The [AllowCreateObject](#) property specifies the combination of keys that allows the user to create objects on the surface.

The following samples show how you can programmatically add a new element:

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

```
With Surface1
  With .Elements
    .Add "new 1"
    .Add "new 1",24,24
  End With
End With
```

End With

## VB6

```
With Surface1
    With .Elements
        .Add "new 1"
        .Add "new 1",24,24
    End With
End With
```

## VB.NET

```
With Exsurface1
    With .Elements
        .Add("new 1")
        .Add("new 1",24,24)
    End With
End With
```

## VB.NET for /COM

```
With AxSurface1
    With .Elements
        .Add("new 1")
        .Add("new 1",24,24)
    End With
End With
```

## C++

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

    #import <ExSurface.dll>
    using namespace EXSURFACELib;
*/
```

```
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)->GetControlUnknown();
EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();
var_Elements->Add("new 1",vtMissing,vtMissing);
var_Elements->Add("new 1",long(24),long(24));
```

## C++ Builder

```
Exsurfacelib_tlb::IElementsPtr var_Elements = Surface1->Elements;
var_Elements->Add(TVariant("new 1"),TNoParam(),TNoParam());
var_Elements->Add(TVariant("new 1"),TVariant(24),TVariant(24));
```

## C#

```
exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;
var_Elements.Add("new 1",null,null);
var_Elements.Add("new 1",24,24);
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
var var_Elements = Surface1.Elements;
var_Elements.Add("new 1",null,null);
var_Elements.Add("new 1",24,24);
</SCRIPT>
```

## C# for /COM

```
EXSURFACELib.Elements var_Elements = axSurface1.Elements;
var_Elements.Add("new 1",null,null);
var_Elements.Add("new 1",24,24);
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Elements;
    anytype var_Elements;
    ;

    super();

    var_Elements = exsurface1.Elements(); com_Elements = var_Elements;
    com_Elements.Add("new 1");
    com_Elements.Add("new
1",COMVariant::createFromInt(24),COMVariant::createFromInt(24));
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do
begin
    with Elements do
    begin
        Add('new 1',Nil,Nil);
        Add('new 1',TObject(24),TObject(24));
    end;
end
```

## Delphi (standard)

```
with Surface1 do
begin
    with Elements do
    begin
        Add('new 1',Null,Null);
        Add('new 1',OleVariant(24),OleVariant(24));
    end;
end
```

## VFP



```
with thisform.Surface1
  with .Elements
    .Add("new 1")
    .Add("new 1",24,24)
  endwhile
endwith
```

## dBASE Plus

```
local oSurface,var_Elements

oSurface = form.Activex1.nativeObject
var_Elements = oSurface.Elements
  var_Elements.Add("new 1")
  var_Elements.Add("new 1",24,24)
```

## XBasic (Alpha Five)

```
Dim oSurface as P
Dim var_Elements as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
var_Elements = oSurface.Elements
  var_Elements.Add("new 1")
  var_Elements.Add("new 1",24,24)
```

## Visual Objects

```
local var_Elements as IElements

var_Elements := oDCOCX_Exontrol1:Elements
  var_Elements.Add("new 1",nil,nil)
  var_Elements.Add("new 1",24,24)
```

## PowerBuilder

```
OleObject oSurface,var_Elements
```

```
oSurface = ole_1.Object
```

```
var_Elements = oSurface.Elements
```

```
var_Elements.Add("new 1")
```

```
var_Elements.Add("new 1",24,24)
```

## Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Variant voElements
```

```
Get ComElements to voElements
```

```
Handle hoElements
```

```
Get Create (RefClass(cComElements)) to hoElements
```

```
Set pvComObject of hoElements to voElements
```

```
Get ComAdd of hoElements "new 1" Nothing Nothing to Nothing
```

```
Get ComAdd of hoElements "new 1" 24 24 to Nothing
```

```
Send Destroy to hoElements
```

```
End_Procedure
```

## XBase++

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

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

```
PROCEDURE Main
```

```
LOCAL oForm
```

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

```
LOCAL oElements
```

```
LOCAL oSurface
```

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

```
oForm:drawingArea:clipChildren := .T.
```

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

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(,, {10,60},{610,370} )

oElements := oSurface:Elements()
oElements:Add("new 1")
oElements:Add("new 1",24,24)

oForm:Show()
DO WHILE nEvent != xbeP_Quit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

# method Elements.Clear ()

Removes all objects in a collection.

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

Use the Clear method to remove all elements from the surface. Use the [Remove](#) method to remove a specific element from the Elements collection. The [RemoveElement](#) event occurs once the element has been removed from the [Elements](#) collection. Use the [RemoveSelection](#) method to remove the selected elements. Removing an element removes the incoming and outgoing links as well. Use the RemoveElement event to release any extra data associated with the element.

# property Elements.Count as Long

Returns the number of elements in the collection.

Type	Description
Long	A Long expression that specifies the number of elements on the surface.

The Count property specifies the number of elements in the Elements collection. The [Item](#) property accesses the element giving its identifier.

The following VB sample enumerates the elements on the surface:

```
Dim e As Variant
For Each e In Surface1.Elements
    Debug.Print e.ID
Next
```

# method Elements.Insert ([Caption as Variant], [Parent as Variant], [Position as Variant])

Inserts a child Element object to the collection and returns a reference to the newly created object.

Type	Description
Caption as Variant	A String expression that specifies the caption to be displayed on the element. The Caption property specifies the value of the <a href="#">Caption</a> parameter. Use the CaptionAlign property to align the element's caption.
Parent as Variant	A Long, String or Numeric expression that specifies the identifier of the parent element, or a reference to the parent element. The <a href="#">Parent</a> property specifies the element's parent. The <a href="#">ID</a> property indicates the element's identifier.
Position as Variant	A Long expression that specifies the position of the element in the parent's children collection. If missing, the element is added at the end of the parent's children list
Return	Description
<a href="#">Element</a>	A Reference to the newly inserted element.

Use the Insert method to insert programmatically a child element. The [Add](#) method adds programmatically a new element to the surface. Use the [InsertControl](#) method to insert programmatically a child element that hosts an inner ActiveX control. The control fires the [AddElement](#) event once a new element is added to the Elements collection. The [AutoSize](#) property specifies whether the element's size if computed automatically based on its content. While the AutoSize property is True, the element is not resizable. Use the [Width](#) / [Height](#) property to specify the size of the element. The [Expanded](#) property expands or collapse the parent element. Use the [IndentX](#) / [IndentY](#) property to specify the indentation between child and parent elements.

The following samples show how you can programmatically add a child element, or create a hierarchy:

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

```
With Surface1
  With .Elements
    .Add("Root").ID = "rootID"
    .Insert "Child 1","rootID"
```

```
.Insert("Child 2","rootID").ID = "childID"  
.Insert "Child 3","rootID"  
.Insert "Sub-Child 1.2","childID"  
.Insert "Sub-Child 2.2","childID"
```

End With

End With

## VB6

With Surface1

With .Elements

```
.Add("Root").ID = "rootID"  
.Insert "Child 1","rootID"  
.Insert("Child 2","rootID").ID = "childID"  
.Insert "Child 3","rootID"  
.Insert "Sub-Child 1.2","childID"  
.Insert "Sub-Child 2.2","childID"
```

End With

End With

## VB.NET

With Exsurface1

With .Elements

```
.Add("Root").ID = "rootID"  
.Insert("Child 1","rootID")  
.Insert("Child 2","rootID").ID = "childID"  
.Insert("Child 3","rootID")  
.Insert("Sub-Child 1.2","childID")  
.Insert("Sub-Child 2.2","childID")
```

End With

End With

## VB.NET for /COM

With AxSurface1

With .Elements

```
.Add("Root").ID = "rootID"
```

```
.Insert("Child 1","rootID")
.Insert("Child 2","rootID").ID = "childID"
.Insert("Child 3","rootID")
.Insert("Sub-Child 1.2","childID")
.Insert("Sub-Child 2.2","childID")
```

End With

End With

## C++

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

```
#import <ExSurface.dll>  
using namespace EXSURFACELib;
```

```
*/  
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-  
> GetControlUnknown();  
EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();  
var_Elements->Add("Root",vtMissing,vtMissing)->PutID("rootID");  
var_Elements->Insert("Child 1","rootID",vtMissing);  
var_Elements->Insert("Child 2","rootID",vtMissing)->PutID("childID");  
var_Elements->Insert("Child 3","rootID",vtMissing);  
var_Elements->Insert("Sub-Child 1.2","childID",vtMissing);  
var_Elements->Insert("Sub-Child 2.2","childID",vtMissing);
```

## C++ Builder

```
Exsurfacedlib_tlb::IElementsPtr var_Elements = Surface1->Elements;  
var_Elements->Add(TVariant("Root"),TNoParam(),TNoParam())-  
> set_ID(TVariant("rootID"));  
var_Elements->Insert(TVariant("Child 1"),TVariant("rootID"),TNoParam());  
var_Elements->Insert(TVariant("Child 2"),TVariant("rootID"),TNoParam())-  
> set_ID(TVariant("childID"));  
var_Elements->Insert(TVariant("Child 3"),TVariant("rootID"),TNoParam());
```



```
var_Elements->Insert(TVariant("Sub-Child 1.2"),TVariant("childID"),TNoParam());  
var_Elements->Insert(TVariant("Sub-Child 2.2"),TVariant("childID"),TNoParam());
```

## C#

```
exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;  
var_Elements.Add("Root",null,null).ID = "rootID";  
var_Elements.Insert("Child 1","rootID",null);  
var_Elements.Insert("Child 2","rootID",null).ID = "childID";  
var_Elements.Insert("Child 3","rootID",null);  
var_Elements.Insert("Sub-Child 1.2","childID",null);  
var_Elements.Insert("Sub-Child 2.2","childID",null);
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"  
id="Surface1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
var var_Elements = Surface1.Elements;  
var_Elements.Add("Root",null,null).ID = "rootID";  
var_Elements.Insert("Child 1","rootID",null);  
var_Elements.Insert("Child 2","rootID",null).ID = "childID";  
var_Elements.Insert("Child 3","rootID",null);  
var_Elements.Insert("Sub-Child 1.2","childID",null);  
var_Elements.Insert("Sub-Child 2.2","childID",null);  
</SCRIPT>
```

## C# for /COM

```
EXSURFACELib.Elements var_Elements = axSurface1.Elements;  
var_Elements.Add("Root",null,null).ID = "rootID";  
var_Elements.Insert("Child 1","rootID",null);  
var_Elements.Insert("Child 2","rootID",null).ID = "childID";  
var_Elements.Insert("Child 3","rootID",null);  
var_Elements.Insert("Sub-Child 1.2","childID",null);
```

```
var_Elements.Insert("Sub-Child 2.2","childID",null);
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Element,com_Elements;
    anytype var_Element,var_Elements;
    ;

    super();

    var_Elements = exsurface1.Elements(); com_Elements = var_Elements;
    var_Element = COM::createFromObject(com_Elements.Add("Root"));
    com_Element = var_Element;
    com_Element.ID("rootID");
    com_Elements.Insert("Child 1","rootID");
    var_Element = COM::createFromObject(com_Elements.Insert("Child 2","rootID"));
    com_Element = var_Element;
    com_Element.ID("childID");
    com_Elements.Insert("Child 3","rootID");
    com_Elements.Insert("Sub-Child 1.2","childID");
    com_Elements.Insert("Sub-Child 2.2","childID");
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do
begin
    with Elements do
    begin
        Add('Root',Nil,Nil).ID := 'rootID';
        Insert('Child 1','rootID',Nil);
        Insert('Child 2','rootID',Nil).ID := 'childID';
        Insert('Child 3','rootID',Nil);
        Insert('Sub-Child 1.2','childID',Nil);
        Insert('Sub-Child 2.2','childID',Nil);
    end;
end;
```

```
end;  
end
```

## Delphi (standard)

```
with Surface1 do  
begin  
  with Elements do  
  begin  
    Add('Root',Null,Null).ID := 'rootID';  
    Insert('Child 1','rootID',Null);  
    Insert('Child 2','rootID',Null).ID := 'childID';  
    Insert('Child 3','rootID',Null);  
    Insert('Sub-Child 1.2','childID',Null);  
    Insert('Sub-Child 2.2','childID',Null);  
  end;  
end
```

## VFP

```
with thisform.Surface1  
  with .Elements  
    .Add("Root").ID = "rootID"  
    .Insert("Child 1","rootID")  
    .Insert("Child 2","rootID").ID = "childID"  
    .Insert("Child 3","rootID")  
    .Insert("Sub-Child 1.2","childID")  
    .Insert("Sub-Child 2.2","childID")  
  endwith  
endwith
```

## dBASE Plus

```
local oSurface,var_Element,var_Element1,var_Elements  
  
oSurface = form.ActiveX1.nativeObject  
var_Elements = oSurface.Elements  
  // var_Elements.Add("Root").ID = "rootID"
```

```

var_Element = var_Elements.Add("Root")
with (oSurface)
    TemplateDef = [Dim var_Element]
    TemplateDef = var_Element
    Template = [var_Element.ID = "rootID"]
endwith
var_Elements.Insert("Child 1","rootID")
// var_Elements.Insert("Child 2","rootID").ID = "childID"
var_Element1 = var_Elements.Insert("Child 2","rootID")
with (oSurface)
    TemplateDef = [Dim var_Element1]
    TemplateDef = var_Element1
    Template = [var_Element1.ID = "childID"]
endwith
var_Elements.Insert("Child 3","rootID")
var_Elements.Insert("Sub-Child 1.2","childID")
var_Elements.Insert("Sub-Child 2.2","childID")

```

## XBasic (Alpha Five)

```

Dim oSurface as P
Dim var_Element as P
Dim var_Element1 as P
Dim var_Elements as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
var_Elements = oSurface.Elements
' var_Elements.Add("Root").ID = "rootID"
var_Element = var_Elements.Add("Root")
oSurface.TemplateDef = "Dim var_Element"
oSurface.TemplateDef = var_Element
oSurface.Template = "var_Element.ID = \"rootID\""

var_Elements.Insert("Child 1","rootID")
' var_Elements.Insert("Child 2","rootID").ID = "childID"
var_Element1 = var_Elements.Insert("Child 2","rootID")

```

```
oSurface.TemplateDef = "Dim var_Element1"  
oSurface.TemplateDef = var_Element1  
oSurface.Template = "var_Element1.ID = \"childID\""  
  
var_Elements.Insert("Child 3","rootID")  
var_Elements.Insert("Sub-Child 1.2","childID")  
var_Elements.Insert("Sub-Child 2.2","childID")
```

## Visual Objects

```
local var_Elements as IElements
```

```
var_Elements := oDCOCX_Exontrol1.Elements  
var_Elements.Add("Root",nil,nil):ID := "rootID"  
var_Elements.Insert("Child 1","rootID",nil)  
var_Elements.Insert("Child 2","rootID",nil):ID := "childID"  
var_Elements.Insert("Child 3","rootID",nil)  
var_Elements.Insert("Sub-Child 1.2","childID",nil)  
var_Elements.Insert("Sub-Child 2.2","childID",nil)
```

## PowerBuilder

```
OleObject oSurface,var_Elements  
  
oSurface = ole_1.Object  
var_Elements = oSurface.Elements  
var_Elements.Add("Root").ID = "rootID"  
var_Elements.Insert("Child 1","rootID")  
var_Elements.Insert("Child 2","rootID").ID = "childID"  
var_Elements.Insert("Child 3","rootID")  
var_Elements.Insert("Sub-Child 1.2","childID")  
var_Elements.Insert("Sub-Child 2.2","childID")
```

## Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voElements

Get ComElements to voElements

Handle hoElements

Get Create (RefClass(cComElements)) to hoElements

Set pvComObject of hoElements to voElements

Variant voElement

Get ComAdd of hoElements "Root" Nothing Nothing to voElement

Handle hoElement

Get Create (RefClass(cComElement)) to hoElement

Set pvComObject of hoElement to voElement

Set ComID of hoElement to "rootID"

Send Destroy to hoElement

Get **ComInsert** of hoElements "Child 1" "rootID" Nothing to Nothing

Variant voElement1

Get **ComInsert** of hoElements "Child 2" "rootID" Nothing to voElement1

Handle hoElement1

Get Create (RefClass(cComElement)) to hoElement1

Set pvComObject of hoElement1 to voElement1

Set ComID of hoElement1 to "childID"

Send Destroy to hoElement1

Get **ComInsert** of hoElements "Child 3" "rootID" Nothing to Nothing

Get **ComInsert** of hoElements "Sub-Child 1.2" "childID" Nothing to Nothing

Get **ComInsert** of hoElements "Sub-Child 2.2" "childID" Nothing to Nothing

Send Destroy to hoElements

End\_Procedure

## XBase++

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

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

```
PROCEDURE Main
```

```
LOCAL oForm
```

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

```
LOCAL oElements
```

## LOCAL oSurface

```
oForm := XbpDialog():new( AppDesktop() )
oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(,, {10,60},{610,370} )
```

```
oElements := oSurface:Elements()
oElements:Add("Root"):ID := "rootID"
oElements:Insert("Child 1","rootID")
oElements:Insert("Child 2","rootID"):ID := "childID"
oElements:Insert("Child 3","rootID")
oElements:Insert("Sub-Child 1.2","childID")
oElements:Insert("Sub-Child 2.2","childID")
```

```
oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
```

```
RETURN
```

## method Elements.InsertControl ([Control as Variant], [License as Variant], [Parent as Variant], [Position as Variant])

Inserts a child Element object ( that hosts another control inside ) to the collection and returns a reference to the newly created object.

Type	Description
Control as Variant	A string expression that can be formatted as follows: a prog ID, a CLSID, a URL, a reference to an Active document , a fragment of HTML. The <a href="#">Control</a> property returns the value of Control parameter.
License as Variant	A string expression that indicates the runtime license key for the component being inserted, if required. Only, the vendor of the component you are going to use is able to give you such of runtime license, so please contact the control's vendor for such of key. Your development license key is not compatible with the runtime license key, so it can't be used here. The <a href="#">License</a> property specifies the runtime-license of the control.
Parent as Variant	A Long, String or Numeric expression that specifies the identifier of the parent element or a reference to the parent element. The <a href="#">Parent</a> property specifies the element's parent.
Position as Variant	The position of the element in the parent's children collection. If missing, the element is added at the end of the children list. The <a href="#">ChildPosition</a> property returns the element's position in the parent children collection.
Return	Description
<a href="#">Element</a>	An Element object that hosts the specified control.

The control supports ActiveX hosting, or in other words, any element can host another inside controls. The InsertControl property adds a new Element object with the [Type](#) set on exElementHostControl, that hosts an ActiveX control. The inside ActiveX control is specified by the Control and License parameters. The [Object](#) property returns a reference to newly created control. The ExSurface control fires the [OLEEvent](#) event if an inside ActiveX control fires an event. Use the [ElementFormat](#) property to specify the area where the inner control is displayed.

The Control parameter must be formatted in one of the following ways:

- A ProgID such as "Exontrol.Grid"



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

The following samples shows how you can host a Command button:

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

```
With Surface1
  With .Elements
    With .InsertControl("Forms.CommandButton.1")
      .ElementFormat = """"check""":18,""client""""
      .Object.Caption = "command"
      .ShowCheckBox = True
      .Height = 48
      .Width = 128
    End With
  End With
End With
```

## VB6

```
With Surface1
  With .Elements
    With .InsertControl("Forms.CommandButton.1")
      .ElementFormat = """"check""":18,""client""""
      .Object.Caption = "command"
      .ShowCheckBox = True
      .Height = 48
      .Width = 128
    End With
  End With
End With
```

## VB.NET

```
With Exsurface1
```

```
With .Elements
```

```
With .InsertControl("Forms.CommandButton.1")
```

```
.ElementFormat = ""check":18,"client""
```

```
.Object.Caption = "command"
```

```
.ShowCheckBox = True
```

```
.Height = 48
```

```
.Width = 128
```

```
End With
```

```
End With
```

```
End With
```

## VB.NET for /COM

```
With AxSurface1
```

```
With .Elements
```

```
With .InsertControl("Forms.CommandButton.1")
```

```
.ElementFormat = ""check":18,"client""
```

```
.Object.Caption = "command"
```

```
.ShowCheckBox = True
```

```
.Height = 48
```

```
.Width = 128
```

```
End With
```

```
End With
```

```
End With
```

## C++

```
/*
```

Copy and paste the following directives to your header file as it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control Library'

```
#import <ExSurface.dll>
```

```
using namespace EXSURFACELib;
```

```
*/
```

```
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)->GetControlUnknown();
```

```

EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();
EXSURFACELib::IElementPtr var_Element = var_Elements-
> InsertControl("Forms.CommandButton.1",vtMissing,vtMissing,vtMissing);
var_Element->PutElementFormat(L"\check\":18,\"client\");
/*

```

Copy and paste the following directives to your header file as it defines the namespace 'MSForms' for the library: 'Microsoft Forms 2.0 Object Library'

```

#import <FM20.DLL>
*/
((MSForms::ICommandButtonPtr)(var_Element->GetObject()))-
> PutCaption(L"command");
var_Element->PutShowCheckBox(VARIANT_TRUE);
var_Element->PutHeight(48);
var_Element->PutWidth(128);

```

## C++ Builder

```

Exsurfacedlib_tlb::IElementsPtr var_Elements = Surface1->Elements;
Exsurfacedlib_tlb::IElementPtr var_Element = var_Elements-
> InsertControl(TVariant("Forms.CommandButton.1"),TNoParam(),TNoParam(),TNoParam());

var_Element->ElementFormat = L"\check\":18,\"client\");
(IDispatch*)var_Element->Object->Caption = L"command";
var_Element->ShowCheckBox = true;
var_Element->Height = 48;
var_Element->Width = 128;

```

## C#

```

exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;
exontrol.EXSURFACELib.Element var_Element =
var_Elements.InsertControl("Forms.CommandButton.1",null,null,null);
var_Element.ElementFormat = "\check\":18,\"client\");
// Add 'Microsoft Forms 2.0 Object Library' reference to your project.

```

```
(var_Element.Object as MSForms.CommandButton).Caption = "command";  
var_Element.ShowCheckBox = true;  
var_Element.Height = 48;  
var_Element.Width = 128;
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"  
id="Surface1"></OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
  var var_Elements = Surface1.Elements;  
  var var_Element =  
var_Elements.InsertControl("Forms.CommandButton.1",null,null,null);  
  var_Element.ElementFormat = "\"check\":18,\"client\"";  
  var_Element.Object.Caption = "command";  
  var_Element.ShowCheckBox = true;  
  var_Element.Height = 48;  
  var_Element.Width = 128;  
</SCRIPT>
```

## C# for /COM

```
EXSURFACELib.Elements var_Elements = axSurface1.Elements;  
EXSURFACELib.Element var_Element =  
var_Elements.InsertControl("Forms.CommandButton.1",null,null,null);  
var_Element.ElementFormat = "\"check\":18,\"client\"";  
// Add 'Microsoft Forms 2.0 Object Library' reference to your project.  
(var_Element.Object as MSForms.CommandButton).Caption = "command";  
var_Element.ShowCheckBox = true;  
var_Element.Height = 48;  
var_Element.Width = 128;
```

## X++ (Dynamics Ax 2009)

```
public void init()
```

```

{
  COM com_Element,com_Elements,com_Object;
  anytype var_Element,var_Elements,var_Object;
  ;

  super();

  var_Elements = exsurface1.Elements(); com_Elements = var_Elements;
  var_Element = com_Elements.InsertControl("Forms.CommandButton.1");
  com_Element = var_Element;
  com_Element.ElementFormat("\check\":18,\"client\");
  var_Object = COM::createFromObject(com_Element.Object()); com_Object =
  var_Object;
  com_Object.Caption("command");
  com_Element.ShowCheckBox(true);
  com_Element.Height(48);
  com_Element.Width(128);
}

```

## Delphi 8 (.NET only)

```

with AxSurface1 do
begin
  with Elements do
  begin
    with InsertControl('Forms.CommandButton.1',Nil,Nil,Nil) do
    begin
      ElementFormat := "'check':18,\"client\"";
      (Object as MSForms.CommandButton).Caption := 'command';
      ShowCheckBox := True;
      Height := 48;
      Width := 128;
    end;
  end;
end

```

## Delphi (standard)

```

with Surface1 do
begin
  with Elements do
  begin
    with InsertControl('Forms.CommandButton.1',Null,Null,Null) do
    begin
      ElementFormat := "'check':18,'client'";
      (IUnknown(Object) as MSForms_TLB.CommandButton).Caption := 'command';
      ShowCheckBox := True;
      Height := 48;
      Width := 128;
    end;
  end;
end
end

```

## VFP

```

with thisform.Surface1
  with .Elements
    with InsertControl("Forms.CommandButton.1")
      .ElementFormat =
        "" + chr(34) + "check" + chr(34) + ":18," + chr(34) + "client" + chr(34) + ""
      .Object.Caption = "command"
      .ShowCheckBox = .T.
      .Height = 48
      .Width = 128
    endwith
  endwith
endwith

```

## dBASE Plus

```

local oSurface,var_Element,var_Elements

oSurface = form.ActiveX1.nativeObject
var_Elements = oSurface.Elements
var_Element = var_Elements.InsertControl("Forms.CommandButton.1")
var_Element.ElementFormat = "" + ["] + "check" + ["] + ":18," + ["] + "client" +

```

```
["] + ""
```

```
var_Element.Object.Caption = "command"  
var_Element.ShowCheckBox = true  
var_Element.Height = 48  
var_Element.Width = 128
```

## XBasic (Alpha Five)

```
Dim oSurface as P  
Dim var_Element as P  
Dim var_Elements as P
```

```
oSurface = topparent:CONTROL_ACTIVEX1.activex  
var_Elements = oSurface.Elements  
var_Element = var_Elements.InsertControl("Forms.CommandButton.1")  
var_Element.ElementFormat = "\"check\":18,\"client\""  
var_Element.Object.Caption = "command"  
var_Element.ShowCheckBox = .t.  
var_Element.Height = 48  
var_Element.Width = 128
```

## Visual Objects

```
local var_Element as IElement  
local var_Elements as IElements
```

```
var_Elements := oDCOCX_Exontrol1:Elements  
var_Element := var_Elements.InsertControl("Forms.CommandButton.1",nil,nil,nil)  
var_Element:ElementFormat := "" + CHR(34) + "check" + CHR(34) + ":18," +  
CHR(34) + "client" + CHR(34) + ""
```

**// Generate Source for 'Microsoft Forms 2.0 Object Library' server from  
Tools\Automation Server...**

```
ICommandButton{var_Element:Object}:Caption := "command"  
var_Element:ShowCheckBox := true  
var_Element:Height := 48  
var_Element:Width := 128
```

# PowerBuilder

```
OleObject oSurface,var_Element,var_Elements

oSurface = ole_1.Object
var_Elements = oSurface.Elements
  var_Element = var_Elements.InsertControl("Forms.CommandButton.1")
    var_Element.ElementFormat = "" + CHAR(34) + "check" + CHAR(34) + ":18," +
CHAR(34) + "client" + CHAR(34) + ""
    var_Element.Object.Caption = "command"
    var_Element.ShowCheckBox = true
    var_Element.Height = 48
    var_Element.Width = 128
```

# Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Variant voElements
  Get ComElements to voElements
  Handle hoElements
  Get Create (RefClass(cComElements)) to hoElements
  Set pvComObject of hoElements to voElements
  Variant voElement
  Get ComInsertControl of hoElements "Forms.CommandButton.1" Nothing
Nothing Nothing to voElement
  Handle hoElement
  Get Create (RefClass(cComElement)) to hoElement
  Set pvComObject of hoElement to voElement
  Set ComElementFormat of hoElement to ""check":18,"client""
  Variant voCommandButton
  Get ComObject of hoElement to voCommandButton
  Handle hoCommandButton
  Get Create (RefClass(cComCommandButton)) to hoCommandButton
  Set pvComObject of hoCommandButton to voCommandButton
```



```
Set ComCaption of hoCommandButton to "command"
Send Destroy to hoCommandButton
Set ComShowCheckBox of hoElement to True
Set ComHeight of hoElement to 48
Set ComWidth of hoElement to 128
Send Destroy to hoElement
Send Destroy to hoElements
End_Procedure
```

## XBase++

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

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oElement
  LOCAL oElements
  LOCAL oSurface

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oSurface := XbpActiveXControl():new( oForm:drawingArea )
  oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
  oSurface:create(,, {10,60},{610,370} )

  oElements := oSurface:Elements()
  oElement := oElements:InsertControl("Forms.CommandButton.1")
  oElement:ElementFormat := "" + CHR(34) + "check" + CHR(34) + ":18," +
CHR(34) + "client" + CHR(34) + ""
  oElement:Object():Caption := "command"
  oElement:ShowCheckBox := .T.
```

oElement:Height := 48

oElement:Width := 128

oForm:Show()

DO WHILE nEvent != xbeP\_Quit

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

    oXbp:handleEvent( nEvent, mp1, mp2 )

ENDDO

RETURN

# property Elements.Item (ID as Variant) as Element

Returns a specific Element of the Elements collection, giving its identifier.

Type	Description
ID as Variant	A Long, String or Numeric expression that defines the element's unique identifier.
<a href="#">Element</a>	An Element object being retrieved.

The Item property accesses the element giving its identifier. The [Count](#) property specifies the number of elements in the Elements collection. The [ID](#) property specifies the element's unique identifier.

The following VB sample enumerates the elements on the surface:

```
Dim e As Variant
For Each e In Surface1.Elements
    Debug.Print e.ID
Next
```

# method Elements.Remove (ID as Variant)

Removes a specific member from the Elements collection, giving its identifier or reference.

Type	Description
ID as Variant	A Long, String or Numeric expression that specifies the identifier of the element to be removed, or a reference to the Element to be removed.

Use the Remove method to remove a specific element from the Elements collection. The [RemoveElement](#) event occurs once the element has been removed from the [Elements](#) collection. Use the [RemoveSelection](#) method to remove the selected elements. Use the [Clear](#) method to remove all elements from the surface. Removing an element removes the incoming and outgoing links as well. Use the RemoveElement event to release any extra data associated with the element. Use the [Visible](#) property to hide a specific element.

# ExDataObject object

Defines the object that contains OLE drag and drop information.

Name	Description
<a href="#">Clear</a>	Deletes the contents of the ExDataObject object.
<a href="#">Files</a>	Returns an ExDataObjectFiles collection, which in turn contains a list of all filenames used by an ExDataObject object.
<a href="#">GetData</a>	Returns data from an ExDataObject object in the form of a variant.
<a href="#">GetFormat</a>	Returns a value indicating whether an item in the ExDataObject object matches a specified format.
<a href="#">SetData</a>	Inserts data into an ExDataObject object using the specified data format.

# method ExDataObject.Clear ()

Deletes the contents of the DataObject object.

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

The Clear method can be called only for drag sources. The [OleDragDrop](#) event notifies your application that the user drags some data on the control.

# property ExDataObject.Files as ExDataObjectFiles

Returns a DataObjectFiles collection, which in turn contains a list of all filenames used by a DataObject object.

Type	Description
<a href="#">ExDataObjectFiles</a>	An ExDataObjectFiles object that contains a list of filenames used in OLE drag and drop operations

The Files property is valid only if the format of the clipboard data is exCFFiles. The [OleDragDrop](#) event notifies your application that the user drags some data on the control.

# method ExDataObject.GetData (Format as Integer)

Returns data from a DataObject object in the form of a variant.

Type	Description
Format as Integer	An <a href="#">exClipboardFormatEnum</a> expression that defines the data's format
Return	Description
Variant	A Variant value that contains the ExDataObject's data in the given format

Use GetData property to retrieve the clipboard's data that has been dragged to the control. It's possible for the GetData and [SetData](#) methods to use data formats other than [exClipboardFormatEnum](#) , including user-defined formats registered with Windows via the RegisterClipboardFormat() API function. The GetData method always returns data in a byte array when it is in a format that it is not recognized. Use the [Files](#) property to retrieves the filenames if the format of data is exCFFiles



# method ExDataObject.GetFormat (Format as Integer)

Returns a value indicating whether the ExDataObject's data is of the specified format.

Type	Description
Format as Integer	A constant or value that specifies a clipboard data format like described in <a href="#">exClipboardFormatEnum</a> enum.
Return	Description
Boolean	A boolean value that indicates whether the ExDataObject's data is of specified format.

Use the GetFormat property to verify if the ExDataObject's data is of a specified clipboard format. The GetFormat property retrieves True, if the ExDataObject's data format matches the given data format.

# method ExDataObject.SetData ([Value as Variant], [Format as Variant])

Inserts data into a ExDataObject object using the specified data format.

Type	Description
Value as Variant	A data that is going to be inserted to ExDataObject object.
Format as Variant	A constant or value that specifies the data format, as described in <a href="#">exClipboardFormatEnum</a> enum

Use SetData property to insert data for OLE drag and drop operations. Use the [Files](#) property is you are going to add new files to the clipboard data. The [OleDragDrop](#) event notifies your application that the user drags some data on the control.

# ExDataObjectFiles object

The ExDataObjectFiles contains a collection of filenames. The ExDataObjectFiles object is used in OLE Drag and drop events. In order to get the list of files used in drag and drop operations you have to use the [Files](#) property.

Name	Description
<a href="#">Add</a>	Adds a filename to the Files collection
<a href="#">Clear</a>	Removes all file names in the collection.
<a href="#">Count</a>	Returns the number of file names in the collection.
<a href="#">Item</a>	Returns an specific file name.
<a href="#">Remove</a>	Removes an specific file name.

# method ExDataObjectFiles.Add (FileName as String)

Adds a filename to the Files collection

Type	Description
FileName as String	A string expression that indicates a filename.

Use Add method to add your files to ExDataObject object. The [OleStartDrag](#) event notifies your application that the user starts dragging items.

# method ExDataObjectFiles.Clear ()

Removes all file names in the collection.

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

Use the Clear method to remove all filenames from the collection.

# property ExDataObjectFiles.Count as Long

Returns the number of file names in the collection.

Type	Description
Long	A long value that indicates the count of elements into collection.

You can use "for each" statements if you are going to enumerate the elements into ExDataObjectFiles collection.

# property ExDataObjectFiles.Item (Index as Long) as String

Returns a specific file name given its index.

Type	Description
Index as Long	A long expression that indicates the filename's index
String	A string value that indicates the filename

# method ExDataObjectFiles.Remove (Index as Long)

Removes a specific file name given its index into collection.

Type	Description
Index as Long	A long expression that indicates the index of filename into collection.

Use [Clear](#) method to remove all filenames.



# HitTest object

The HitTest object determines the element and the hit-test code. The [HitTestFromPoint](#) property returns the hit-test object from the cursor. The HitTest object supports the following properties and methods:

Name	Description
<a href="#">Element</a>	Specifies the element object.
<a href="#">HitTestCode</a>	Specifies the hit-test code.
<a href="#">HitTestKey</a>	Specifies the hit-test key.

# property HitTest.Element as Element

Specifies the element object.

Type	Description
<a href="#">Element</a>	An Element object

The Element property specifies the Element from the cursor. The [HitTestCode](#) property specifies the code where the cursor hovers the element. The [HitTestKey](#) property indicates the key associated to the part of the element from the cursor.

# property `HitTest.HitTestCode` as `HitTestCodeEnum`

Specifies the hit-test code.

Type	Description
<a href="#">HitTestCodeEnum</a>	A <code>HitTestCodeEnum</code> expression that specifies the code of the part where the cursor is over the element.

The `HitTestCode` property specifies the code where the cursor hovers the element. The [Element](#) property specifies the Element from the cursor. The [HitTestKey](#) property indicates the key associated to the part of the element from the cursor.

The `HitTestCode` may indicate one of the following parts of the element:

- **exHitTestMargin**, indicates the border of the element. The [HitTestKey](#) property returns nothing.
- **exHitTestStatus**, indicates the status part of the element. The [HitTestKey](#) property returns nothing.
- **exHitTestClient**, indicates the element's background ( empty ). The [HitTestKey](#) property returns nothing.
- **exHitTestPicture**, indicates any icon/picture on from the element. The [HitTestKey](#) property specifies the identifier of the icon/picture from the cursor.
- **exHitTestCaption**, indicates the element's [Caption](#). The [HitTestKey](#) property specifies the element's caption.
- **exHitTestExtraCaption**, indicates the element's [ExtraCaption](#). The [HitTestKey](#) property specifies the element's extra-caption.
- **exHitTestCheckBox**, indicates the element's [checkbox](#). The [HitTestKey](#) property specifies the element's [Checked](#) property.
- **exHitTestGlyph**, indicates the element's expand/collapse glyph. The [HitTestKey](#) property returns nothing.

# property HitTest.HitTestKey as Variant

Specifies the hit-test key.

Type	Description
Variant	A String expression that determines the key associated with the part of the element from the cursor.

The HitTestKey property indicates the key associated to the part of the element from the cursor. The [Element](#) property specifies the Element from the cursor. The [HitTestCode](#) property specifies the code where the cursor hovers the element.

The HitTestKey may returns one of the following values, based on the [HitTestCode](#) property as listed:

- nothing, if [HitTestCode](#) property is exHitTestMargin, exHitTestStatus, exHitTestClient and exHitTestGlyph.
- the identifier of the icon/picture from the cursor if [HitTestCode](#) property is exHitTestPicture.
- the element's caption if [HitTestCode](#) property is exHitTestCaption.
- the element's extra-caption if [HitTestCode](#) property is exHitTestExtraCaption.
- element's [Checked](#) property, if [HitTestCode](#) property is exHitTestCheckBox.

# Link object

The Link object defines a link between two elements on the surface. The link starts from [ElementFrom](#) element and ends on [ElementTo](#) element. The Link object supports the following properties and methods:

Name	Description
<a href="#">AllowControlPoint</a>	Indicates the control points of the link, the user can use to customize the link.
<a href="#">ArrowColor</a>	Gets or sets a value that indicates the link's arrow color.
<a href="#">ArrowFrameColor</a>	Customizes the color to show the frame of the arrow.
<a href="#">ArrowSize</a>	Gets or sets the size to show the arrow for specified link.
<a href="#">Caption</a>	Gets or sets a value that indicates the HTML caption to be displayed on the link.
<a href="#">CaptionAlign</a>	Indicates the alignment of the link's caption.
<a href="#">Color</a>	Gets or sets a value that indicates the link's color.
<a href="#">CustomPath</a>	Specifies the link's custom path.
<a href="#">ElementFrom</a>	Specifies the element where the link starts from.
<a href="#">ElementTo</a>	Specifies the element where the link ends into.
<a href="#">EndPos</a>	Specifies where the link ends on the target/to element.
<a href="#">EndUpdateLink</a>	Adds programmatically updated properties of the link to undo/redo queue.
<a href="#">ID</a>	Specifies the link's unique identifier.
<a href="#">ShowDir</a>	Shows or hides the link's direction.
<a href="#">ShowLinkType</a>	Specifies how the link shows from source to target element.
<a href="#">StartPos</a>	Specifies where the link starts on the source/from element.
<a href="#">StartUpdateLink</a>	Starts changing properties of the link, so EndUpdateLink method adds programmatically updated properties to undo/redo queue.
<a href="#">Style</a>	Specifies the link's style.
<a href="#">ToolTip</a>	Gets or sets a value (tooltip) that's displayed once the cursor hovers the link.
<a href="#">ToolTipTitle</a>	Gets or sets a value (title) that's displayed once the cursor hovers the link.

[UserData](#)

Indicates any extra data associated with the link.

[Visible](#)

Shows or hides the link.

[Width](#)

Gets or sets a value that indicates the link's width.

# property Link.AllowControlPoint as LinkControlPointEnum

Indicates the control points of the link, the user can use to customize the link.

Type	Description
<a href="#">LinkControlPointEnum</a>	A LinkControlPointEnum expression that indicates the control points of the link, the user can use to customize the link.

The AllowControlPoint property defines the control points for an individual link, the user can use to customize the link. The AllowControlPoint property is similar with the control's [AllowLinkControlPoint](#) property, excepts that it is applied to a link only. For instance, exNoControlPoint specifies that the link displays no control points, so the user can not customize the link's path. The link's control points are displayed only if the control is not locked (control's DesignMode property is not exDesignLock The [LayoutStartChanging\(exLinkControlPoint\)](#) / [LayoutEndChanging\(exLinkControlPoint\)](#) events as soon as user starts / ends changing the link's control points. The [CustomPath](#) property specifies the link's custom path, as a string of x,y proportions separated by comma. The CustomPath property contains the proportions of link's control-points, as a "x,y,x,y,x,y,...". The x, y are proportions of link's control-points relative to the start/end points of the link. The 0,0 indicates the link's start point, while 1,1 indicates the link's end point. For instance, "0.5,0,0.5,1" defines the link to go from start (0,0) to (0.5,0), then (0.5,1), and finally to the end (1,1)

# property Link.ArrowColor as Color

Gets or sets a value that indicates the link's arrow color.

Type	Description
Color	A Color expression that defines the color to show the arrow of the link. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used to paint the part. Use the <a href="#">Add</a> method to add new skins to the control. In other words, you can use the EBN objects to define a different type of arrow.

By default, the ArrowColor property is -1, which indicates that the control's [LinksArrowColor](#) property indicates the color to show the arrow or direction of the link. The ArrowColor property specifies the color to show the direction of the particular link. The [Color](#) property specifies the color to show the entire link. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. The [Width](#) property specifies the size of the link and so the size of the arrow. The [Visible](#) property indicates whether the link is visible or hidden.



# property Link.ArrowFrameColor as Color

Customizes the color to show the frame of the arrow.

Type	Description
Color	A Color expression to show the arrow's frame

By default, the ArrowFrameColor property is -1 which indicates that [LinksArrowFrameColor](#) property specifies the color of the arrow's frame. The ArrowFrameColor property specifies the color to show the arrow's frame for a particular link. The [LinksArrowFrameColor](#) property specifies the color to show the default frame of the arrow. Use the [LinksColor](#) property to define the color to show all links on the surface. The [Color](#) property specifies the color for an individual link. The [LinksArrowColor](#) property specifies the color to show the arrow of the links. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [LinksArrowSize](#) property specifies the size to show the arrow for links.

# property Link.ArrowSize as Long

Gets or sets the size to show the arrow for specified link.

Type	Description
Long	A long expression that specifies the size of the arrow

By default, the ArrowSize property is -1 which indicates that [LinksArrowSize](#) property controls the size of the arrow. The ArrowSize property specifies the size to show the arrow for a particular link. The [LinksArrowSize](#) property specifies the size to show the arrow for links. Use the [LinksColor](#) property to define the color to show all links on the surface. The [Color](#) property specifies the color for an individual link. The [LinksArrowColor](#) property specifies the color to show the arrow of the links. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [LinksArrowFrameColor](#) property specifies the color to show the default frame of the arrow.

# property Link.Caption as String

Gets or sets a value that indicates the HTML caption to be displayed on the link.

Type	Description
String	A String expression that defines the HTML caption to be displayed on the link.

Use the Caption property to define the label or caption to be displayed on the link. The [CaptionAlign](#) property specifies the alignment of the caption on the link. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

The Caption property supports the following HTML elements:

- `<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.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using `<a ;exp=>` or `<a ;e64=>` anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu</a>`" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY" string encodes the "`<fgcolor 808080>show lines<a>-</a></fgcolor>`" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "`<solidline><b>Header</b></solidline>`"

<br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **<font face;size> ... </font>** displays portions of text with a different font and/or different size. For instance, the "**<font Tahoma;12>bit</font>**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**<font ;12>bit</font>**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **<br>** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&quot;**; ( " ) and **&#number;**; ( the character with specified code ), For instance, the **&#8364;** displays the EUR

character. The & ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display `<b>bold</b>` in HTML caption you can use `&lt;b&gt;bold&lt;/b&gt;`;

- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `<font face;size>` to define a smaller or a larger font to be displayed. For instance: "Text with `<font ;7><off 6>`subscript" displays the text such as: Text with subscript The "Text with `<font ;7><off -6>`superscript" displays the text such as: Text with subscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `<font>` HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "`<font ;18><gra FFFFFFFF;1;1>`gradient-center`</gra></font>`" generates the following picture:

gradient-center

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

outlined

- **`<sha rrggbb;width;offset> ... </sha>`** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `<font>` HTML tag can be used to define the height of the font. For instance the "`<font ;31><sha>`shadow`</sha></font>`" generates the following picture:

shadow

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

# outline anti-aliasing

# property Link.CaptionAlign as AlignmentEnum

Indicates the alignment of the link's caption.

Type	Description
<a href="#">AlignmentEnum</a>	An AlignmentEnum expression that defines the alignment of the caption on the link.

By default, the CaptionAlign property is CenterAlignment. Use the CaptionAlign property to align the caption on the start or end element of the link. Use the [Caption](#) property to define the label or caption to be displayed on the link. The [ElementFrom](#) property defines the element where the link starts. The [ElementTo](#) property defines the element where the link ends.

# property Link.Color as Color

Gets or sets a value that indicates the link's color.

Type	Description
Color	A Color expression that specifies the color to show the link.

By default, the Color property is -1, which indicates that the [LinksColor](#) property specifies the color to show the links. The Color property specifies the color to show the entire link. The [ArrowColor](#) property specifies the color to show the direction of the particular link. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. The [Width](#) property specifies the size of the link and so the size of the arrow. The [Visible](#) property indicates whether the link is visible or hidden. The [ShowLinksColor](#) property specifies the color to show the incoming, outgoing or collapsed links.



# property Link.CustomPath as Variant

Specifies the link's custom path.

Type	Description
Variant	A string expression that define the link's custom path.

The CustomPath property specifies the link's custom path, as a string of x,y proportions separated by comma. The CustomPath property contains the proportions of link's control-points, as a "x,y,x,y,x,y,...". The x, y are proportions of link's control-points relative to the start/end points of the link. The 0,0 indicates the link's start point, while 1,1 indicates the link's end point. For instance, "0.5,0,0.5,1" defines the link to go from start (0,0) to (0.5,0), then (0.5,1), and finally to the end (1,1). The [AllowLinkControlPoint](#) property indicates the control points of the link, the user can use to customize the link's path. The [AllowControlPoint](#) property defines the control points for an individual link, the user can use to customize the link.

# property Link.ElementFrom as Element

Specifies the element where the link starts from.

Type	Description
<a href="#">Element</a>	An Element object that specifies where the link starts.

The ElementFrom property specifies where the link starts. The [ElementTo](#) property specifies the element where the link ends. The [StartPos/EndPos](#) properties indicates where on the starting element links starts and where on the ending elements the link ends. The [ShowDir](#) property specifies whether the direction of the link is shown or hidden.

# property Link.ElementTo as Element

Specifies the element where the link ends into.

Type	Description
<a href="#">Element</a>	An Element object where the link ends.

The ElementTo property specifies the element where the link ends. The [ElementFrom](#) property specifies where the link starts. The [StartPos/EndPos](#) properties indicates where on the starting element links starts and where on the ending elements the link ends. The [ShowDir](#) property specifies whether the direction of the link is shown or hidden.

# property Link.EndPos as AlignmentEnum

Specifies where the link ends on the target/to element.

Type	Description
<a href="#">AlignmentEnum</a>	An AlignmentEnum expression that specifies where on the ending element the link ends.

The [StartPos](#)/EndPos properties indicate where on the starting element the link starts and where on the ending elements the link ends. The [ShowDir](#) property specifies whether the direction of the link is shown or hidden. The [ElementTo](#) property specifies the element where the link ends. The [ElementFrom](#) property specifies where the link starts.

## method `Link.EndUpdateLink (StartUpdateLink as Long)`

Adds programmatically updated properties of the link to undo/redo queue.

Type	Description
StartUpdateLink as Long	A long expression that indicates the result of the <a href="#">StartUpdateLink</a> property

The [StartUpdateLink](#)/EndUpdateLink methods record and add changes of the current link to the control's Undo/Redo queue. You can use the [StartBlockUndoRedo](#) / [EndBlockUndoRedo](#) methods to group multiple Undo/Redo operations into a single-block. The [AllowUndoRedo](#) property specifies whether the control supports undo/redo operations for objects (links, links, ...). No entry is added to the Undo/Redo queue if no property is changed for the current link. Each call of the [StartUpdateLink](#) must be succeeded by a EndUpdateLink call. The [UndoListAction](#) property lists the Undo actions that can be performed in the chart. The [RedoListAction](#) property lists the Redo actions that can be performed in the chart.

The [StartUpdateLink](#)/EndUpdateLink methods can record changes for the following properties only:

- [ElementFrom](#), specifies the element where the link starts from
- [ElementTo](#), specifies the element where the link ends into
- [ID](#), specifies the link's unique identifier
- [Style](#), specifies the link's style
- [Color](#), specifies the link's color
- [Width](#), specifies the link's width
- [Visible](#), shows or hides the link
- [StartPos](#), specifies where the link starts on the source/from element
- [EndPos](#), specifies where the link ends on the target/to element
- [ShowLinkType](#), specifies how the link shows from source to target element
- [ShowDir](#), shows or hides the link's direction/arrow
- [ArrowColor](#), defines the link's arrow color
- [UserData](#), associates any extra data associated with the link
- [Caption](#), gets or sets a value that indicates the HTML caption to be displayed on the link
- [ToolTip](#), gets or sets a value (tooltip) that's displayed once the cursor hovers the link
- [ToolTipTitle](#), gets or sets a value (title) that's displayed once the cursor hovers the link
- [AllowControlPoint](#), indicates the control points of the link, the user can use to customize the link
- [CustomPath](#), specifies the link's custom path

The Undo/Redo records show as:

- **"UpdateLink;LINKID"**, indicates that one or more properties of the element has been

updated, using the [StartUpdateLink](#) / EndUpdateLink methods

within the [UndoListAction/RedoListAction](#) result

# property Link.ID as Variant

Specifies the link's unique identifier.

Type	Description
Variant	A Long, String or Numeric expression that defines the unique identifier of the link.

By default, the control automatically generates an unique identifier for each link. You can use the ID property to define your id for the link. The control fires the [CreateLink](#) event when the user adds at runtime a link between two elements. The [AllowLinkObjects](#) property specifies the keys combination to let the user links two elements on the surface, the [ElementFrom](#) property specifies the element where the link starts, where the [ElementTo](#) property specifies ending element of the link. Prior to CreateLink event the control fires the [AddLink](#) event that indicates that the link has been added to the [Links](#) collection. The [AllowLink](#) event occurs when user links two elements to specify whether the link is allowed.

# property Link.ShowDir as Boolean

Shows or hides the link's direction.

Type	Description
Boolean	A Boolean expression that specifies whether the link's direction is shown or hidden.

The ShowDir property specifies whether the arrow of the link is shown or hidden. The [LinksShowDir](#) property specifies whether the direction for all links are shown or hidden. The [ArrowColor](#) property specifies the color to show the direction of the particular link. The [Width](#) property specifies the size of the link and so the size of the arrow. The [Visible](#) property indicates whether the link is visible or hidden. The [StartPos](#)/[EndPos](#) properties indicates where on the starting element links starts and where on the ending elements the link ends.



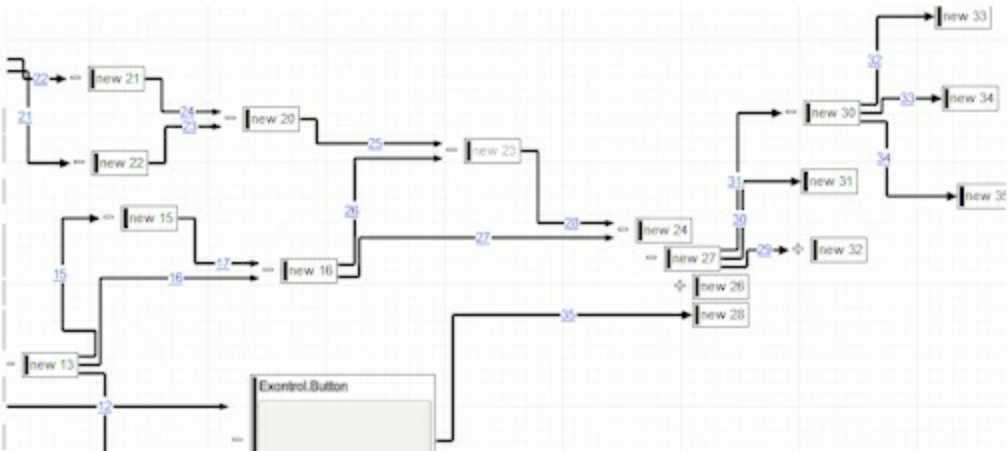
# property Link.ShowLinkType as ShowLinkTypeEnum

Specifies how the link shows from source to target element.

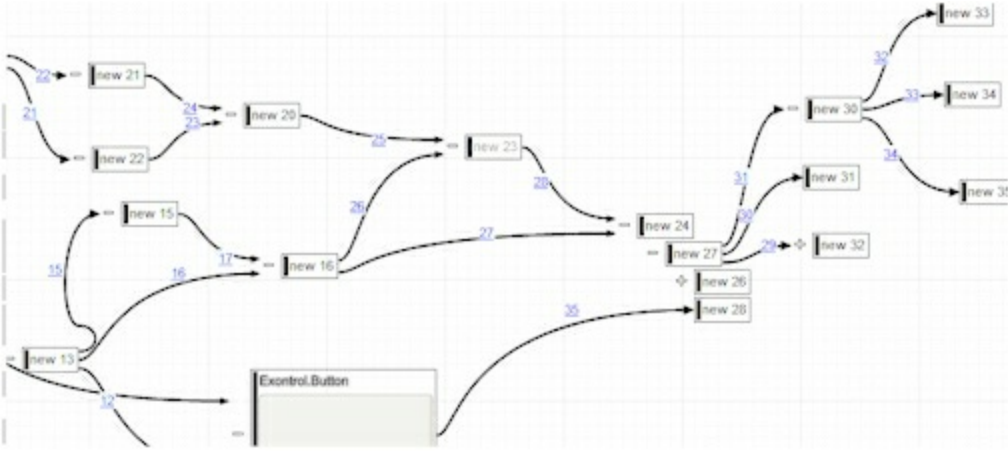
Type	Description
<a href="#">ShowLinkTypeEnum</a>	A ShowLinkTypeEnum expression that defines the type of the link to be show between elements.

By default, the ShowLinkType property is -1, which specifies that the control's [ShowLinksType](#) property indicates the type of the links to be shown between elements. The ShowLinkType property specifies a different type of link between two elements. The [Color](#) property specifies the color to show the entire link. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. The [Width](#) property specifies the size of the link and so the size of the arrow. The [Visible](#) property indicates whether the link is visible or hidden. The [Style](#) property defines the style of the line to be shown on the link.

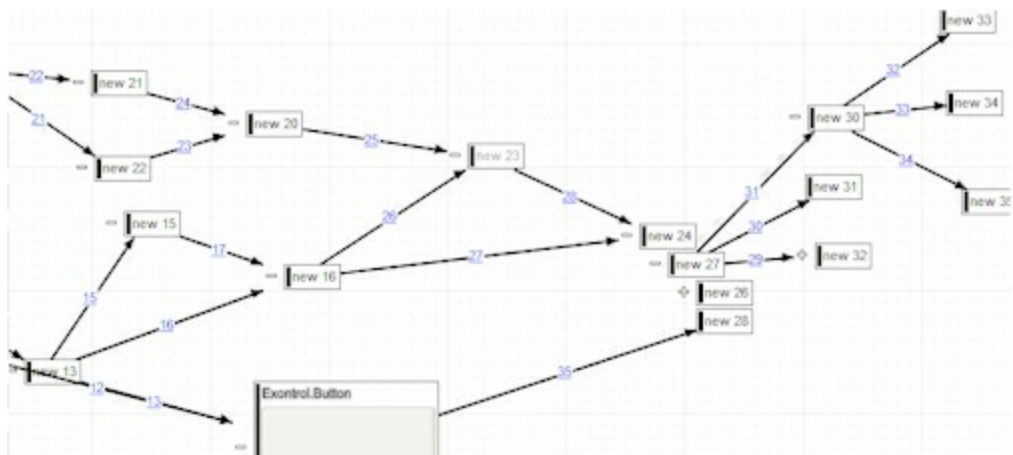
The following screen shot shows the exLinkRectangular type:



The following screen shot shows the exLinkRound type:



The following screen shot shows the exLinkDirect type:



# property Link.StartPos as AlignmentEnum

Specifies where the link starts on the source/from element.

Type	Description
<a href="#">AlignmentEnum</a>	An AlignmentEnum expression that specifies where on the starting element the link starts.

The StartPos/[EndPos](#) properties indicates where on the starting element links starts and where on the ending elements the link ends. The [ShowDir](#) property specifies whether the direction of the link is shown or hidden. The [ElementTo](#) property specifies the element where the link ends. The [ElementFrom](#) property specifies where the link starts.

## property Link.StartUpdateLink as Long

Starts changing properties of the link, so EndUpdateLink method adds programmatically updated properties to undo/redo queue.

Type	Description
Long	A Long expression that specifies the handle to be passed to <a href="#">EndUpdateLink</a> so the updated properties of the bar are added to the Undo/Redo queue of the chart, so they can be used in undo/redo operations.

The StartUpdateLink/[EndUpdateLink](#) methods record and add changes of the current link to the control's Undo/Redo queue. You can use the [StartBlockUndoRedo](#) / [EndBlockUndoRedo](#) methods to group multiple Undo/Redo operations into a single-block. The [AllowUndoRedo](#) property specifies whether the control supports undo/redo operations for objects (links, links, ...). No entry is added to the Undo/Redo queue if no property is changed for the current link. Each call of the StartUpdateLink must be succeeded by a [EndUpdateLink](#) call. The [UndoListAction](#) property lists the Undo actions that can be performed in the chart. The [RedoListAction](#) property lists the Redo actions that can be performed in the chart.

The StartUpdateLink/[EndUpdateLink](#) methods can record changes for the following properties only:

- [ElementFrom](#), specifies the element where the link starts from
- [ElementTo](#), specifies the element where the link ends into
- [ID](#), specifies the link's unique identifier
- [Style](#), specifies the link's style
- [Color](#), specifies the link's color
- [Width](#), specifies the link's width
- [Visible](#), shows or hides the link
- [StartPos](#), specifies where the link starts on the source/from element
- [EndPos](#), specifies where the link ends on the target/to element
- [ShowLinkType](#), specifies how the link shows from source to target element
- [ShowDir](#), shows or hides the link's direction/arrow
- [ArrowColor](#), defines the link's arrow color
- [UserData](#), associates any extra data associated with the link
- [Caption](#), gets or sets a value that indicates the HTML caption to be displayed on the link
- [ToolTip](#), gets or sets a value (tooltip) that's displayed once the cursor hovers the link
- [ToolTipTitle](#), gets or sets a value (title) that's displayed once the cursor hovers the link
- [AllowControlPoint](#), indicates the control points of the link, the user can use to customize the link
- [CustomPath](#), specifies the link's custom path

The Undo/Redo records show as:

- "**UpdateLink**;LINKID", indicates that one or more properties of the element has been updated, using the StartUpdateLink / [EndUpdateLink](#) methods

within the [UndoListAction](#)/[RedoListAction](#) result

# property Link.Style as LinkStyleEnum

Specifies the link's style.

Type	Description
<a href="#">LinkStyleEnum</a>	A LinkStyleEnum expression that defines the style of the line to be shown between elements.

By default, the Style property is -1, which specifies that the control's [LinksStyle](#) property specifies the style of the line to be shown on links. The Style property indicates the style of the line to be shown on a particular link. The [ShowLinkType](#) property specifies the type of the link to be shown, like rectangular, straight and so on. The [Width](#) property specifies the size of the link and so the size of the arrow. The [Visible](#) property indicates whether the link is visible or hidden. The [Color](#) property specifies the color to show the entire link. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. The [ShowLinksStyle](#) property specifies the style to show the incoming, outgoing or collapsed links.

# property Link.ToolTip as String

Gets or sets a value (tooltip) that's displayed once the cursor hovers the link.

Type	Description
String	A String expression that defines the HTML caption to be shown when the cursor hovers the link.

Use the ToolTip property to define the label or caption to be shown when the cursor hovers the link. The [ToolTipTitle](#) defines the title of the link's tooltip. The [Images](#) method loads icons to be displayed on the control's surface. The [HTMLPicture](#) property loads and assigns a picture to a key to be used on control's surface.

The ToolTip property supports the following HTML elements:

- `<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.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using `<a ;exp=>` or `<a ;e64=>` anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu</a>`" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY" string encodes the "`<fgcolor 808080>show lines<a>-</a></fgcolor>`" The `Decode64Text/Encode64Text` methods of the `eXPrint` can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "`<solidline><b>Header</b></solidline>`"

<br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

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



character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display `<b>bold</b>` in HTML caption you can use `&lt;b&gt;bold&lt;/b&gt;`;

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `<font face;size>` to define a smaller or a larger font to be displayed. For instance: "Text with `<font ;7><off 6>`subscript" displays the text such as: Text with subscript The "Text with `<font ;7><off -6>`superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `<font>` HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "`<font ;18><gra FFFFFFFF;1;1>`gradient-center`</gra></font>`" generates the following picture:

gradient-center

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

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `<font>` HTML tag can be used to define the height of the font. For instance the "`<font ;31><sha>`shadow`</sha></font>`" generates the following picture:

shadow

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

# outline anti-aliasing

# property Link.ToolTipTitle as String

Gets or sets a value (title) that's displayed once the cursor hovers the link.

Type	Description
String	A String expression that defines the title of the link's tooltip

The ToolTipTitle defines the title of the link's tooltip. Use the [ToolTip](#) property to define the label or caption to be shown when the cursor hovers the link.

# property Link.UserData as Variant

Indicates any extra data associated with the link.

Type	Description
Variant	A VARIANT expression that defines any extra-data you can associate with the link.

The UserData property specifies any extra-data associated with the link. The control fires the [AddLink](#) event when a new link is added to the surface. The [RemoveLink](#) event notifies your application once a link is removed form the surface.

# property Link.Visible as Boolean

Shows or hides the link.

Type	Description
Boolean	A Boolean expression that specifies whether the link is visible or hidden.

By default, the Visible property is True. Use the Visible property to hide a particular link. The Visible property indicates whether the link is visible or hidden. The [ShowLinks](#) property specifies whether the control show or hide the links on the surface. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. A link between two elements is visible, if both element are visible. Use the [Visible](#) property to specify whether an element is visible or hidden. Use the [ShowLinksOnCollapse](#) property to show the links between an element and collapsed elements.

# property Link.Width as Long

Gets or sets a value that indicates the link's width.

Type	Description
Long	A Long expression that specifies the size of the link.

By default, the Width property of the link is -1, which specifies that the control's [LinksWidth](#) property indicates the width of the link. The Width property specifies the width of a particular link. The Width property specifies the size of the link and so the size of the arrow. The [Color](#) property specifies the color to show the entire link. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. The [Visible](#) property indicates whether the link is visible or hidden. The [ShowLinksWidth](#) property specifies the width to show the incoming, outgoing or collapsed links.

# Links object

The Links collection holds the links to be shown on the surface. The Links collection can be accessed through the control's [Links](#) property. The Links collection supports the following properties and methods:

Name	Description
<a href="#">Add</a>	Adds a Link 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>	Returns the number of elements in the collection.
<a href="#">Item</a>	Returns a specific Link of the Links collection, giving its identifier.
<a href="#">Remove</a>	Removes a specific member from the Links collection, giving its identifier or reference.

# method Links.Add (From as Element, To as Element, [ID as Variant])

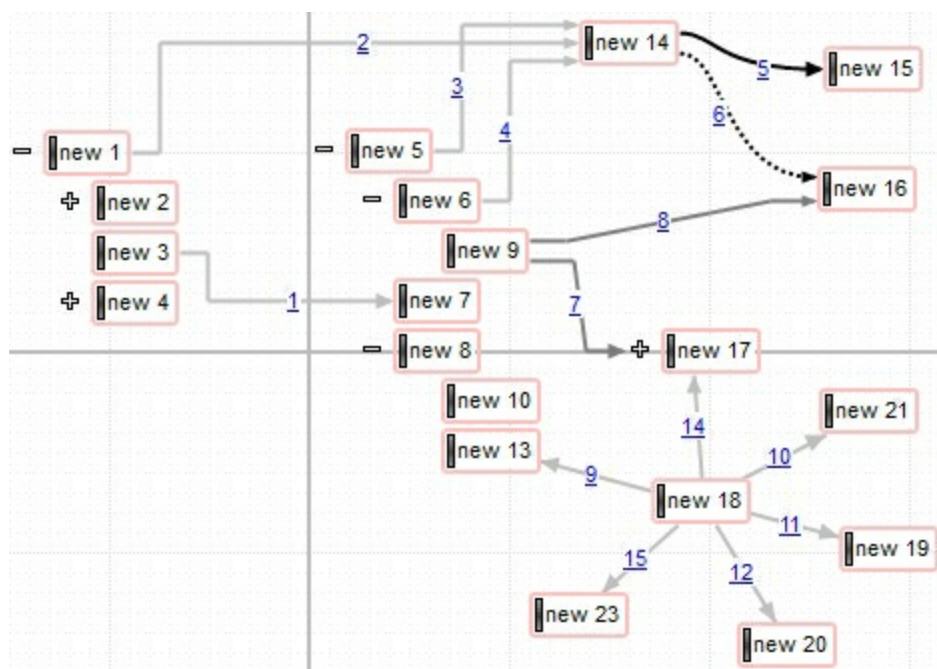
Adds a Link object to the collection and returns a reference to the newly created object.

Type	Description
From as <a href="#">Element</a>	An Element object where the link starts. The <a href="#">ElementFrom</a> property of the Link indicates the element where the link starts.
To as <a href="#">Element</a>	An Element object where the link ends. The <a href="#">ElementTo</a> property of the Link indicates the element where the link ends.
ID as Variant	A Long, String or Numeric expression that indicates the unique identifier of the link. If missing, the control will automatically generate an unique identifier. The <a href="#">ID</a> property specifies the link's identifier.
Return	Description
<a href="#">Link</a>	A Link object being created.

.Use the Add method to programmatically add new link to the surface. The [AddLink](#) event notifies your application once a new link is added to the Links collection. Calling programmatically the Add method **does NOT** fire the [CreateLink](#) or [AllowLink](#) events. The [Remove](#) method removes a link from the surface. The [Clear](#) method clears all the links on the surface. The [StartPos/EndPos](#) properties indicates where on the starting element links starts and where on the ending elements the link ends. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [OutgoingLinks](#) property returns a safe array of outgoing links ( links that starts from the element ). The [IncomingLinks](#) property returns a safe array of incoming links ( links that ends on the element ). The [AllowLinkObjects](#) property specifies the combination of keys that allows the user to link the objects.

The following screen shot shows the surface with different type of links:





The order of the events when the user links two elements at runtime is:

- [LayoutStartChanging](#)(exLinkObjects), the user clicks on the surface
- [AllowLink](#), occurs to specify whether the link between two elements is possible.
- [AddLink](#), adds the new link to the Links collection
- [CreateLink](#), the user ends creating the link
- [LayoutEndChanging](#)(exLinkObjects), the user un-clicks the surface

The following samples show how you can add programmatically a link:

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

```
With Surface1
  With .Elements
    .Add "Element <sha ;;0>A"
    .Add "Element <sha ;;0>B",96,24
  End With
  With .Links
    .Add Surface1.Elements.item(1),Surface1.Elements.item(2)
  End With
End With
```

### VB6

```
With Surface1
  With .Elements
    .Add "Element <sha ;;0>A"
```

```
.Add "Element <sha ;;0> B",96,24
End With
With .Links
    .Add Surface1.Elements.item(1),Surface1.Elements.item(2)
End With
End With
```

## VB.NET

```
With Exsurface1
    With .Elements
        .Add("Element <sha ;;0> A")
        .Add("Element <sha ;;0> B",96,24)
    End With
    With .Links
        .Add(Exsurface1.Elements.get_item(1),Exsurface1.Elements.get_item(2))
    End With
End With
```

## VB.NET for /COM

```
With AxSurface1
    With .Elements
        .Add("Element <sha ;;0> A")
        .Add("Element <sha ;;0> B",96,24)
    End With
    With .Links
        .Add(AxSurface1.Elements.item(1),AxSurface1.Elements.item(2))
    End With
End With
```

## C++

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

```

#import <ExSurface.dll>
using namespace EXSURFACELib;
*/
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
>GetControlUnknown();
EXSURFACELib::IElementsPtr var_Elements = spSurface1->GetElements();
    var_Elements->Add("Element <sha ;;0>A",vtMissing,vtMissing);
    var_Elements->Add("Element <sha ;;0>B",long(96),long(24));
EXSURFACELib::ILinksPtr var_Links = spSurface1->GetLinks();
    var_Links->Add(spSurface1->GetElements()->Getitem(long(1)),spSurface1-
>GetElements()->Getitem(long(2)),vtMissing);

```

## C++ Builder

```

Exsurfacedlib_tlb::IElementsPtr var_Elements = Surface1->Elements;
    var_Elements->Add(TVariant("Element <sha ;;0>A"),TNoParam(),TNoParam());
    var_Elements->Add(TVariant("Element <sha ;;0>B"),TVariant(96),TVariant(24));
Exsurfacedlib_tlb::ILinksPtr var_Links = Surface1->Links;
    var_Links->Add(Surface1->Elements->get_item(TVariant(1)),Surface1->Elements-
>get_item(TVariant(2)),TNoParam());

```

## C#

```

exontrol.EXSURFACELib.Elements var_Elements = exsurface1.Elements;
    var_Elements.Add("Element <sha ;;0>A",null,null);
    var_Elements.Add("Element <sha ;;0>B",96,24);
exontrol.EXSURFACELib.Links var_Links = exsurface1.Links;
    var_Links.Add(exsurface1.Elements[1],exsurface1.Elements[2],null);

```

## JavaScript

```

<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">

```

```

var var_Elements = Surface1.Elements;
    var_Elements.Add("Element <sha ;;0>A",null,null);
    var_Elements.Add("Element <sha ;;0>B",96,24);
var var_Links = Surface1.Links;
    var_Links.Add(Surface1.Elements.item(1),Surface1.Elements.item(2),null);
</SCRIPT>

```

## C# for /COM

```

EXSURFACELib.Elements var_Elements = axSurface1.Elements;
    var_Elements.Add("Element <sha ;;0>A",null,null);
    var_Elements.Add("Element <sha ;;0>B",96,24);
EXSURFACELib.Links var_Links = axSurface1.Links;
    var_Links.Add(axSurface1.Elements[1],axSurface1.Elements[2],null);

```

## X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Elements,com_Links;
    anytype var_Elements,var_Links;
    ;

    super();

    var_Elements = exsurface1.Elements(); com_Elements = var_Elements;
    com_Elements.Add("Element <sha ;;0>A");
    com_Elements.Add("Element <sha
;;0>B",COMVariant::createFromInt(96),COMVariant::createFromInt(24));
    var_Links = exsurface1.Links(); com_Links = var_Links;

    com_Links.Add(COM::createFromObject(exsurface1.Elements()).item(COMVariant::creat
}

```

## Delphi 8 (.NET only)

```

with AxSurface1 do
begin
  with Elements do
  begin
    Add('Element <sha ;;0>A',Nil,Nil);
    Add('Element <sha ;;0>B',TObject(96),TObject(24));
  end;
  with Links do
  begin

Add(AxSurface1.Elements.item[TObject(1)],AxSurface1.Elements.item[TObject(2)],Nil);
    end;
  end
end

```

## Delphi (standard)

```

with Surface1 do
begin
  with Elements do
  begin
    Add('Element <sha ;;0>A',Null,Null);
    Add('Element <sha ;;0>B',OleVariant(96),OleVariant(24));
  end;
  with Links do
  begin

Add(Surface1.Elements.item[OleVariant(1)],Surface1.Elements.item[OleVariant(2)],Null);

    end;
  end
end

```

## VFP

```

with thisform.Surface1
  with .Elements
    .Add("Element <sha ;;0>A")
    .Add("Element <sha ;;0>B",96,24)
  endwith

```

```
with .Links
```

```
    .Add(thisform.Surface1.Elements.item(1),thisform.Surface1.Elements.item(2))  
endwith  
endwith
```

## dBASE Plus

```
local oSurface,var_Elements,var_Links
```

```
oSurface = form.Activex1.nativeObject
```

```
var_Elements = oSurface.Elements
```

```
    var_Elements.Add("Element <sha ;;0> A")
```

```
    var_Elements.Add("Element <sha ;;0> B",96,24)
```

```
var_Links = oSurface.Links
```

```
    var_Links.Add(oSurface.Elements.item(1),oSurface.Elements.item(2))
```

## XBasic (Alpha Five)

```
Dim oSurface as P
```

```
Dim var_Elements as P
```

```
Dim var_Links as P
```

```
oSurface = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Elements = oSurface.Elements
```

```
    var_Elements.Add("Element <sha ;;0> A")
```

```
    var_Elements.Add("Element <sha ;;0> B",96,24)
```

```
var_Links = oSurface.Links
```

```
    var_Links.Add(oSurface.Elements.item(1),oSurface.Elements.item(2))
```

## Visual Objects

```
local var_Elements as IElements
```

```
local var_Links as ILinks
```

```
var_Elements := oDCOCX_Exontrol1:Elements
```

```
    var_Elements.Add("Element <sha ;;0> A",nil,nil)
```

```
var_Elements:Add("Element <sha ;;0> B",96,24)
var_Links := oDCOCX_Exontrol1:Links
var_Links:Add(oDCOCX_Exontrol1:Elements:[item,1],oDCOCX_Exontrol1:Elements:
[item,2],nil)
```

## PowerBuilder

```
OleObject oSurface,var_Elements,var_Links

oSurface = ole_1.Object
var_Elements = oSurface.Elements
var_Elements.Add("Element <sha ;;0> A")
var_Elements.Add("Element <sha ;;0> B",96,24)
var_Links = oSurface.Links
var_Links.Add(oSurface.Elements.item(1),oSurface.Elements.item(2))
```

## Visual DataFlex

```
Procedure OnCreate
Forward Send OnCreate
Variant voElements
Get ComElements to voElements
Handle hoElements
Get Create (RefClass(cComElements)) to hoElements
Set pvComObject of hoElements to voElements
    Get ComAdd of hoElements "Element <sha ;;0> A" Nothing Nothing to Nothing
    Get ComAdd of hoElements "Element <sha ;;0> B" 96 24 to Nothing
Send Destroy to hoElements
Variant voLinks
Get ComLinks to voLinks
Handle hoLinks
Get Create (RefClass(cComLinks)) to hoLinks
Set pvComObject of hoLinks to voLinks
    Variant vFrom
        Variant voElements1
        Get ComElements to voElements1
```

Handle hoElements1

Get Create (RefClass(cComElements)) to hoElements1

Set pvComObject of hoElements1 to voElements1

Get Comitem of hoElements1 1 to vFrom

Send Destroy to hoElements1

Variant vTo

Variant voElements2

Get ComElements to voElements2

Handle hoElements2

Get Create (RefClass(cComElements)) to hoElements2

Set pvComObject of hoElements2 to voElements2

Get Comitem of hoElements2 2 to vTo

Send Destroy to hoElements2

Get ComAdd of hoLinks vFrom vTo Nothing to Nothing

Send Destroy to hoLinks

End\_Procedure

## XBase++

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

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

```
PROCEDURE Main
```

```
LOCAL oForm
```

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

```
LOCAL oElements
```

```
LOCAL oLinks
```

```
LOCAL oSurface
```

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

```
oForm:drawingArea:clipChildren := .T.
```

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

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oSurface := XbpActiveXControl():new( oForm:drawingArea )
```

```
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-  
2F9B5155EA4B}*/
```



```
oSurface:create(,, {10,60},{610,370} )
```

```
oElements := oSurface:Elements()
```

```
oElements:Add("Element <sha ;;0>A")
```

```
oElements:Add("Element <sha ;;0>B",96,24)
```

```
oLinks := oSurface:Links()
```

```
oLinks:Add(oSurface:Elements:item(1),oSurface:Elements:item(2))
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

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

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

# method Links.Clear ()

Removes all objects in a collection.

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

Use the Clear method to remove all links from the surface. The [Remove](#) method removes a specific link from the Links collection. The [IncomingLinks](#) property returns a safe array of incoming links ( links that ends on the element ). The [OutgoingLinks](#) property returns a safe array of outgoing links ( links that starts from the element ). The [RemoveLink](#) event notifies your application once a link has been removed from the [Links](#) collection. The [ShowLinks](#) property specifies whether the surface shows or hides the links.

# property Links.Count as Long

Returns the number of elements in the collection.

Type	Description
Long	A Long expression that specifies the number of links on the surface.

The Count property specifies the number of links on the surface. Use the [Add](#) method to programmatically add new link to the surface. The [Remove](#) method removes a link from the surface. The [Clear](#) method clears all the links on the surface. The [StartPos/EndPos](#) properties indicates where on the starting element links starts and where on the ending elements the link ends. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [OutgoingLinks](#) property returns a safe array of outgoing links ( links that starts from the element ). The [IncomingLinks](#) property returns a safe array of incoming links ( links that ends on the element ).

The following VB sample enumerates the links on the surface:

```
Dim I As Variant
For Each I In Surface1.Links
    Debug.Print I.ID
Next
```

# property Links.Item (ID as Variant) as Link

Returns a specific Link of the Links collection, giving its identifier.

Type	Description
ID as Variant	A Long, String or Numeric expression that specifies the identifier of the link to be requested.
<a href="#">Link</a>	A Link object being requested.

Use the Item property to access a link giving its identifier. The [ID](#) property of the Link specifies the identifier of the link. The [Count](#) property specifies the number of links on the surface. Use the [Add](#) method to programmatically add new link to the surface. The [Remove](#) method removes a link from the surface. The [Clear](#) method clears all the links on the surface.

The following VB sample enumerates the links on the surface:

```
Dim I As Variant
For Each I In Surface1.Links
    Debug.Print I.ID
Next
```

# method Links.Remove (ID as Variant)

Removes a specific member from the Links collection, giving its identifier or reference.

Type	Description
ID as Variant	A Long, String or Numeric expression that specifies the identifier of the link to be removed, or a reference to the Link object to e removed.

The Remove method removes a specific link from the Links collection. Use the [Clear](#) method to remove all links from the surface. The [IncomingLinks](#) property returns a safe array of incoming links ( links that ends on the element ). The [OutgoingLinks](#) property returns a safe array of outgoing links ( links that starts from the element ). The [RemoveLink](#) event notifies your application once a link has been removed from the [Links](#) collection. The [Visible](#) property shows or hides a specific link.

# OleEvent object

The OleEvent object holds information about an event fired by an ActiveX control hosted by the element.

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

## property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

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

Use the CountParam property to count the parameters of an OLE event. Use the [Name](#) property to get the parameter name. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when [OleEvent](#) event is fired.

```
Private Sub Surface1_OleEvent(ByVal Element As EXSURFACELibCtl.IElement, ByVal Ev As EXSURFACELibCtl.IOleEvent)
    Debug.Print "Event name:" & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
#import <exsurface.dll>
```

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

        COleVariant vt;
```

```

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

void OnOleEventSurface1(LPDISPATCH Element, LPDISPATCH Ev)
{
    EXSURFACELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXSURFACELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i )
);
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}

```

The `#import` clause is required to get the wrapper classes for `IOleEvent` and `IOleEventParam` objects, that are not defined by the MFC class wizard. The same `#import` statement defines the `EXSURFACELib` namespace that include all objects and types of the control's TypeLibrary. In case your `exsurface.dll` library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

Private Sub AxSurface1_OleEvent(ByVal sender As Object, ByVal e As

```



```

AxEXSURFACELib._ISurfaceEvents_OleEventEvent) Handles AxSurface1.OleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXSURFACELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub

```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

private void AxSurface1_OleEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXSURFACELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}

```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an element ( OleEvent event ):

```

*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
    s = s + "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s

```

# property OleEvent.ID as Long

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

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

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

## property OleEvent.Name as String

Retrieves the original name of the fired event.

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

Use the Name property to get the name of the event. Use the [ID](#) property to specify a specified even by its identifier. Use the [ToString](#) property to display information about fired event such us name, parameters, types and values. Use the [CountParam](#) property to count the parameters of an OLE event. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when [OleEvent](#) event is fired.

```
Private Sub Surface1_OleEvent(ByVal Element As EXSURFACELibCtl.IElement, ByVal Ev As EXSURFACELibCtl.IOleEvent)
    Debug.Print "Event name:" & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
#import <exsurface.dll>
```

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

```

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

void OnOleEventSurface1(LPDISPATCH Element, LPDISPATCH Ev)
{
    EXSURFACELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXSURFACELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i )
);
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}

```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXSURFACELib namespace that include all objects and types of the control's TypeLibrary. In case your exsurface.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

Private Sub AxSurface1_OleEvent(ByVal sender As Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OleEventEvent) Handles AxSurface1.OleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXSURFACELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub

```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

private void AxSurface1_OleEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXSURFACELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}

```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an element ( OleEvent event ):

```

*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
    s = s + "Name: " + ev.Param(i).Name + ", Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s

```



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

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

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

Use the CountParam property to count the parameters of an OLE event. Use the [Name](#) property to get the parameter name. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when [OleEvent](#) event is fired.

```
Private Sub Surface1_OleEvent(ByVal Element As EXSURFACELibCtl.IElement, ByVal Ev As EXSURFACELibCtl.IOleEvent)
    Debug.Print "Event name:" & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
#import <exsurface.dll>

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
```

```

    if ( pv->vt == VT_ERROR )
        return szDefault;

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

void OnOleEventSurface1(LPDISPATCH Element, LPDISPATCH Ev)
{
    EXSURFACELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXSURFACELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i )
);
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}

```

The `#import` clause is required to get the wrapper classes for `IOleEvent` and `IOleEventParam` objects, that are not defined by the MFC class wizard. The same `#import` statement defines the `EXSURFACELib` namespace that include all objects and types of the control's TypeLibrary. In case your `exsurface.dll` library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.



The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
Private Sub AxSurface1_OleEvent(ByVal sender As Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OleEventEvent) Handles AxSurface1.OleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXSURFACELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
private void AxSurface1_OleEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXSURFACELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an element ( OleEvent event ):

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
    s = s + "Name: " + ev.Param(i).Name + ", Value: " + Str(ev.Param(i).Value)
```

```
endfor  
wait window nowait s
```

# property OleEvent.ToString as String

Retrieves information about the event.

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

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

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

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

# OleEventParam object

The OleEventParam holds the name and the value for an event's argument.

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

## property OleEventParam.Name as String

Retrieves the name of the event's parameter.

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

Use the CountParam property to count the parameters of an OLE event. Use the [Name](#) property to get the parameter name. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when [OleEvent](#) event is fired.

```
Private Sub Surface1_OleEvent(ByVal Element As EXSURFACELibCtl.IElement, ByVal Ev As EXSURFACELibCtl.IOleEvent)
    Debug.Print "Event name:" & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
#import <exsurface.dll>
```

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

```

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

void OnOleEventSurface1(LPDISPATCH Element, LPDISPATCH Ev)
{
    EXSURFACELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXSURFACELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i )
);
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}

```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXSURFACELib namespace that include all objects and types of the control's TypeLibrary. In case your exsurface.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

Private Sub AxSurface1_OleEvent(ByVal sender As Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OleEventEvent) Handles AxSurface1.OleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXSURFACELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub

```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

private void AxSurface1_OleEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXSURFACELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}

```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an element ( OleEvent event ):

```

*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
    s = s + "Name: " + ev.Param(i).Name + ", Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s

```

# property OleEventParam.Value as Variant

Specifies the value of the event's parameter.

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

Use the CountParam property to count the parameters of an OLE event. Use the [Name](#) property to get the parameter name. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when [OleEvent](#) event is fired.

```
Private Sub Surface1_OleEvent(ByVal Element As EXSURFACELibCtl.IElement, ByVal Ev As EXSURFACELibCtl.IOleEvent)
    Debug.Print "Event name:" & Ev.Name
    If (Ev.CountParam = 0) Then
        Debug.Print "The event has no arguments."
    Else
        Debug.Print "The event has the following arguments:"
        Dim i As Long
        For i = 0 To Ev.CountParam - 1
            Debug.Print Ev(i).Name; " = " & Ev(i).Value
        Next
    End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an element:

```
#import <exsurface.dll>
```

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



```

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

void OnOleEventSurface1(LPDISPATCH Element, LPDISPATCH Ev)
{
    EXSURFACELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXSURFACELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i )
);
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}

```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXSURFACELib namespace that include all objects and types of the control's TypeLibrary. In case your exsurface.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

Private Sub AxSurface1_OleEvent(ByVal sender As Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OleEventEvent) Handles AxSurface1.OleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXSURFACELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub

```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an element:

```

private void AxSurface1_OleEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXSURFACELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}

```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an element ( OleEvent event ):

```

*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
    s = s + "Name: " + ev.Param(i).Name + ", Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s

```

# Pattern object

The Pattern object can be used to apply a pattern and a frame with different colors on an UI element. For instance, the [StatusPattern](#) property indicates the pattern to be applied on the element's status. The Pattern object supports the following properties:

Name	Description
<a href="#">Color</a>	Specifies the pattern color.
<a href="#">FrameColor</a>	Specifies the pattern's frame color.
<a href="#">Type</a>	Retrieves or sets a value that indicates the pattern to fill the element.

# property Pattern.Color as Color

Specifies the pattern color.

Type	Description
Color	A Color expression that specifies the color to show the pattern.

By default, the Color property is 0 ( black ). The Color property indicates the color to display the pattern. The [Type](#) property indicates the type of the pattern to be shown. The [FrameColor](#) property indicates the color to show the frame, if the exPatternFrame flag is included in the Type property.

# property Pattern.FrameColor as Color

Specifies the pattern's frame color.

Type	Description
Color	A Color expression that specifies the color to show the frame.

By default, the FrameColor property is 0 ( black ). The FrameColor property indicates the color to show the frame, if the exPatternFrame flag is included in the [Type](#) property. The [Type](#) property indicates the type of the pattern to be shown. The [Color](#) property indicates the color to display the pattern.

# property Pattern.Type as PatternEnum

Retrieves or sets a value that indicates the pattern to fill the element.

Type	Description
<a href="#">PatternEnum</a>	A PatternEnum expression that specifies the type of the pattern to fill the element.

By default, the Type property is exPatternEmpty which indicates that no pattern is shown. The Type property indicates the pattern to display on the element. The [Color](#) property indicates the color to display the pattern. The [FrameColor](#) property indicates the color to show the frame, if the exPatternFrame flag is included in the Type property.

# Surface 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: {AC1DF7F4-0919-4364-8167-2F9B5155EA4B}. The object's program identifier is: "Exontrol.Surface". The /COM object module is: "ExSurface.dll"

The Surface objects allows you to put elements and links on a surface. You can use the tool to generate organigrams, diagrams, graphs, flowcharts and so on. The Surface objects supports the following properties and methods:

Name	Description
<a href="#">AlignObjectsToGridLines</a>	Specifies whether the objects are aligned to grid lines.
<a href="#">AllowCreateObject</a>	Specifies the combination of keys that allows the user to create objects on the surface.
<a href="#">AllowInsertObject</a>	Gets or sets a value that specifies whether the user can drag and drop elements to other elements to insert them as child elements.
<a href="#">AllowLinkControlPoint</a>	Indicates the control points of the link, the user can use to customize the link.
<a href="#">AllowLinkObjects</a>	Specifies the combination of keys that allows the user to link the objects.
<a href="#">AllowMoveDescendents</a>	Specifies whether all descendents of the focusing element are moved once the focusing element is moved.
<a href="#">AllowMoveNeighbors</a>	Indicates whether the neighbor elements are shifted once the selection is moved or resized, so they won't intersect the dragging objects.
<a href="#">AllowMoveObject</a>	Specifies the combination of keys that allows the user to move the objects.
<a href="#">AllowMoveSelection</a>	Specifies whether the entire selection is moved once the focusing element is moved.
<a href="#">AllowMoveSurface</a>	Specifies the combination of keys that allows the user to move the surface.
<a href="#">AllowResizeObject</a>	Specifies the combination of keys that allows the user to resize the objects.
<a href="#">AllowResizeSelection</a>	Specifies whether the entire selection is resized once the focusing element is resize.
<a href="#">AllowSelectNothing</a>	Empties the selection when the user clicks outside of the elements.
<a href="#">AllowSelectObject</a>	Specifies the combination of keys that allows the user to

	select objects on the surface.
<a href="#">AllowSelectObjectRect</a>	Specifies the combination of keys that allows the user to select objects on the surface, by dragging a rectangle.
<a href="#">AllowToggleSelectKey</a>	Specifies the combination of keys to select multiple not-contiguously objects.
<a href="#">AllowUndoRedo</a>	Enables or disables the Undo/Redo feature.
<a href="#">AllowZoomSurface</a>	Specifies the combination of keys that allows the user to magnify or shrink the surface.
<a href="#">AllowZoomWheelSurface</a>	Enables or disables zooming the control using the mouse wheel.
<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="#">Arrange</a>	Arranges the elements, starting from giving element, based on the links.
<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="#">AxisColor</a>	Indicates the color to show the axis on the surface.
<a href="#">AxisStyle</a>	Specifies the style to display the axis lines.
<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="#">BorderHeight</a>	Sets or retrieves a value that indicates the border height of the control.
<a href="#">BorderWidth</a>	Sets or retrieves a value that indicates the border width of the control.
<a href="#">CancelLayoutChanging</a>	Cancels the current layout changing operation.
<a href="#">CanRedo</a>	Retrieves a value that indicates whether the surface can perform a Redo operation.
<a href="#">CanUndo</a>	Retrieves a value that indicates whether the surface can perform an Undo operation.
<a href="#">Coord</a>	Specifies the type of coordinates the elements of the surface display in.



<a href="#">CopyTo</a>	Exports the control's view to an EMF file.
<a href="#">DefArrange</a>	Retrieves or sets an option for Arrange method.
<a href="#">DrawPartsOrder</a>	Defines the order of the parts the elements display
<a href="#">EditContextMenuItems</a>	Specifies the control's context menu, while editing the element.
<a href="#">ElementFormat</a>	Specifies the way the control shows the parts of the elements.
<a href="#">ElementFromPoint</a>	Gets the Element object from the cursor.
<a href="#">ElementFromPosition</a>	Gets the Element object from the position.
<a href="#">Elements</a>	Retrieves the control's elements.
<a href="#">Enabled</a>	Enables or disables the control.
<a href="#">EndBlockUndoRedo</a>	Ends recording the UI operations and adds the undo/redo operations as a block, so they all can be restored at once, if Undo method is performed.
<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="#">ExpandLinkedElements</a>	Specifies whether the linked elements are expanded or collapsed.
<a href="#">FitToClient</a>	Resizes or/and moves the entire chart to fit the control's client area.
<a href="#">FocusLink</a>	Gets or sets the focused link
<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="#">GroupUndoRedoActions</a>	Groups the next to current Undo/Redo Actions in a single block.
<a href="#">HideSel</a>	Returns a value that determines whether selected item appears highlighted when a control loses the focus.

<a href="#">HitTestFromPoint</a>	Gets the Element object and the Hit-Test code from the cursor.
<a href="#">Home</a>	Restores the view to the origin.
<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="#">IndentX</a>	Specifies the child elements indentation on x-axis.
<a href="#">IndentY</a>	Specifies the child elements indentation on y-axis.
<a href="#">Layout</a>	Saves or loads the control's layout, such as position, zooming factor, selection, and so on.
<a href="#">LinkFromPoint</a>	Gets the Link object from the cursor.
<a href="#">Links</a>	Retrieves the control's links.
<a href="#">LinksArrowColor</a>	Specifies the color/visual appearance to draw the arrows of the links between the elements.
<a href="#">LinksArrowFrameColor</a>	Specifies the color to show the default frame of the arrow
<a href="#">LinksArrowSize</a>	Specifies the size to show the arrow for links
<a href="#">LinksColor</a>	Specifies the color to draw the links between the elements.
<a href="#">LinksShowDir</a>	Specifies whether the links show or hide the direction/arrow.
<a href="#">LinksStyle</a>	Specifies the style to draw the links between the elements.
<a href="#">LinksWidth</a>	Specifies the width in pixels of the pen to draw the links between the elements.
<a href="#">LoadXML</a>	Loads an XML document from the specified location, using MSXML parser.
<a href="#">MajorGridColor</a>	Indicates the color to show the major grid lines on the surface.
<a href="#">MajorGridHeight</a>	Indicates the height between two consecutive major grid lines.
<a href="#">MajorGridStyle</a>	Specifies the style to display the major grid lines.
	Indicates the width between two consecutive major grid

<a href="#">MajorGridWidth</a>	lines.
<a href="#">MinorGridColor</a>	Indicates the color to show the minor grid lines on the surface.
<a href="#">MinorGridHeight</a>	Indicates the height between two consecutive minor grid lines.
<a href="#">MinorGridStyle</a>	Specifies the style to display the minor grid lines.
<a href="#">MinorGridWidth</a>	Indicates the width between two consecutive minor grid lines.
<a href="#">MoveCorner</a>	Moves or scrolls the surface.
<a href="#">MovePoint</a>	Moves or scrolls the surface, so the cursor aligns to specified corner.
<a href="#">OLEDrag</a>	Causes a component to initiate an OLE drag/drop operation.
<a href="#">OLEDropMode</a>	Returns or sets how a target component handles drop operations
<a href="#">OverviewColor</a>	Specifies the color to show objects outside of the surface's client area.
<a href="#">Picture</a>	Retrieves or sets a graphic to be displayed in the control.
<a href="#">PictureDisplay</a>	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background
<a href="#">PointToPosition</a>	Converts the screen coordinates to surface position.
<a href="#">PositionToPoint</a>	Converts the surface position to screen coordinates.
<a href="#">Redo</a>	Redoes the next action in the surface's Redo queue.
<a href="#">RedoListAction</a>	Lists the Redo actions that can be performed on the surface.
<a href="#">RedoRemoveAction</a>	Removes the first redo actions that can be performed on the surface.
<a href="#">Refresh</a>	Refreses the control.
<a href="#">RemoveSelection</a>	Removes the elements in the selection.
<a href="#">Replacelcon</a>	Adds a new icon, replaces an icon or clears the control's image list.
<a href="#">SaveXML</a>	Saves the control's content as XML document to the specified location, using the MSXML parser.
<a href="#">ScrollPos</a>	Specifies the vertical/horizontal scroll position.

<a href="#">ScrollTo</a>	Scrolls the surface to giving position.
<a href="#">ScrollX</a>	Indicates the x-scrolling position of the surface.
<a href="#">ScrollY</a>	Indicates the y-scrolling position of the surface.
<a href="#">SelCount</a>	Indicates the number of elements being selected on the surface.
<a href="#">SelectAll</a>	Selects all selectable elements in the control.
<a href="#">Selection</a>	Returns or sets a safe array of selected elements on the surface.
<a href="#">SelectObjectColor</a>	Indicates the color to show the selected objects.
<a href="#">SelectObjectColorInactive</a>	Indicates the color to show the selected objects, when the surface is not active/focused.
<a href="#">SelectObjectStyle</a>	Specifies the style to display the selected object.
<a href="#">SelectObjectTextColor</a>	Indicates the color to show the text for selected objects.
<a href="#">SelectObjectTextColorInactive</a>	Indicates the color to show the text for selected objects, when the surface is not active/focused.
<a href="#">SelElement</a>	Gets the element being selected giving its index in the selection.
<a href="#">ShowGridLines</a>	Shows or hides the grid lines in the control.
<a href="#">ShowImageList</a>	Specifies whether the control's image list window is visible or hidden.
<a href="#">ShowLinks</a>	Retrieves or sets a value that indicates whether the links between elements are visible or hidden.
<a href="#">ShowLinksColor</a>	Retrieves or sets a value that indicates the color to display the links based on the user selection.
<a href="#">ShowLinksOnCollapse</a>	Specifies whether the links for collapsed elements are shown or hidden.
<a href="#">ShowLinksStyle</a>	Retrieves or sets a value that indicates the style to display the links based on the user selection.
<a href="#">ShowLinksType</a>	Specifies how the links are displayed between the elements.
<a href="#">ShowLinksWidth</a>	Retrieves or sets a value that indicates the width to display the links based on the user selection.
<a href="#">ShowToolTip</a>	Shows the specified tooltip at given position.
<a href="#">SingleSel</a>	Returns or sets a value that indicates whether the user can select one or more objects.

<a href="#">StartBlockUndoRedo</a>	Starts recording the UI operations as a block of undo/redo operations.
<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="#">ToolBarCaption</a>	Specifies the HTML caption of the giving item in the control's toolbar.
<a href="#">ToolBarFormat</a>	Specifies the CRD format to arrange the buttons inside the control's toolbar.
<a href="#">ToolBarHTMLPicture</a>	Adds or replaces a picture in toolbar's HTML captions.
<a href="#">ToolBarImages</a>	Sets at runtime the toolbar's image list. The Handle should be a handle to an Images List Control.
<a href="#">ToolBarRefresh</a>	Refreshes the control's toolbar.
<a href="#">ToolBarReplacelcon</a>	Adds a new icon, replaces an icon or clears the toolbar's image list.
<a href="#">ToolBarToolTip</a>	Specifies the HTML tooltip of the giving item in the control's toolbar.
<a href="#">ToolBarVisible</a>	Shows or hides control's toolbar.
<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="#">Undo</a>	Performs the last Undo operation.
<a href="#">UndoListAction</a>	Lists the Undo actions that can be performed on the surface.
<a href="#">UndoRedoQueueLength</a>	Gets or sets the maximum number of Undo/Redo actions that may be stored to the surface's queue.
<a href="#">UndoRemoveAction</a>	Removes the last undo actions that can be performed on the surface.
<a href="#">UnselectAll</a>	Unselects all elements in the control.

<a href="#">Version</a>	Retrieves the control's version.
<a href="#">VisualAppearance</a>	Retrieves the control's appearance.
<a href="#">VisualDesign</a>	Invokes the control's VisualAppearance designer.
<a href="#">Zoom</a>	Specifies the current zooming factor of the surface.
<a href="#">ZoomLevels</a>	Specifies the list of zooming factors to be displayed on the control's toolbar.
<a href="#">ZoomMax</a>	Specifies the maximum zooming factor of the surface.
<a href="#">ZoomMin</a>	Specifies the minimum zooming factor of the surface.
<a href="#">ZoomStep</a>	Specifies the step to increase or decrease the zooming factor of the surface, while the user rotates the mouse wheel.

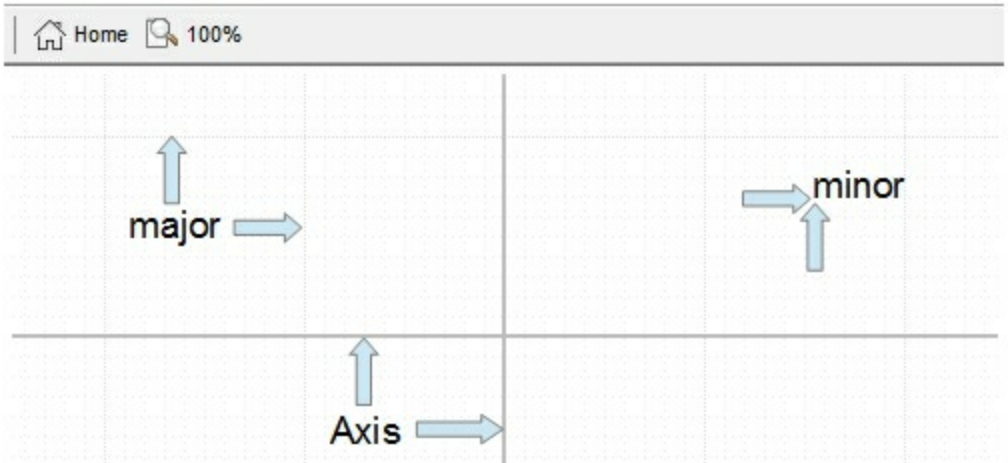
# property Surface.AlignObjectsToGridLines as AlignObjectsToGridLinesEnum

Specifies whether the objects are aligned to grid lines.

Type	Description
<a href="#">AlignObjectsToGridLinesEnum</a>	An AlignObjectsToGridLinesEnum expression that indicates whether the elements on the surface are aligned to none, minor or major grid lines.

By default, the AlignObjectsToGridLines property is exAlignObjectsToNone, which indicates that no alignment is performed, when the elements are shown on the surface. Use the AlignObjectsToGridLines property to align the elements to the grid lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

The following screen shot shows the axis and grid lines:



# property Surface.AllowCreateObject as AllowKeysEnum

Specifies the combination of keys that allows the user to create objects on the surface.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination to allow user creates new elements on the surface.

By default, the AllowCreateObject property exLeftClick + exDbIcIck, which means that a double click on the surface will create new elements. The AllowCreateObject property specifies the combination of keys that allows the user to create objects on the surface. Set the AllowCreateObject property on exDisallow to prevent creating new elements at runtime. Use the [Background](#)(exCreateObjectColor) property to specify color or visual appearance of the creation rectangle when the user creates a new object. The [CreateElement](#) event occurs when the user creates the element on the surface. The control fires the [LayoutStartChanging](#)(exCreateObject) / [LayoutEndChanging](#)(exCreateObject) event when the user creates a new element on the surface.

The order of the events when the user creates the element at runtime is:

- [LayoutStartChanging](#)(exCreateObject), the user clicks on the surface
- [AddElement](#), adds the new element to the Elements collection
- [CreateElement](#), the user ends creating the object
- [LayoutEndChanging](#)(exCreateObject), the user un-clicks the surface

The following screen shot shows the creating rectangle:





# property Surface.AllowInsertObject as Boolean

Gets or sets a value that specifies whether the user can drag and drop elements to other elements to insert them as child elements.

Type	Description
Boolean	A Boolean expression that specifies whether the user can drag and drop elements to other elements to insert them as child elements.

Use the AllowInsertObject property to specify whether the elements can be be dropped over other elements to change its parent or the children list. Use the [AllowChangeParent](#) property to prevent changing the element's Parent at runtime. The [AllowInsertChild](#) property specifies whether other elements can be inserted as child elements of the current element. The [Parent](#) property indicates the element's parent. The [Children](#) property specifies the list of child elements. The control fires the [ParentChangeEvent](#) event when the user changes the element's parent.

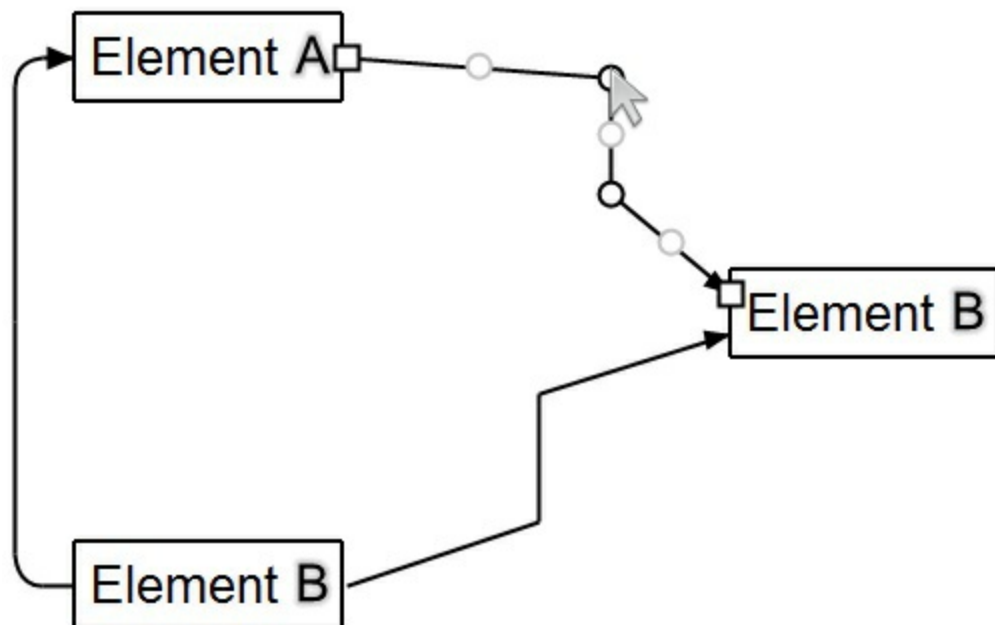
# property Surface.AllowLinkControlPoint as LinkControlPointEnum

Indicates the control points of the link, the user can use to customize the link.

Type	Description
<a href="#">LinkControlPointEnum</a>	A LinkControlPointEnum expression that specifies the control points of the link, the user can use to customize the link.

The AllowLinkControlPoint property indicates the control points of the link, the user can use to customize the link's path. The property is applied to all links (unless the [AllowControlPoint](#) property indicates a different value). For instance, exStartControlPoint | exEndControlPoint specifies that the user can change only the start/end position of the link. The exStartControlPoint and exEndControlPoint points are marked with black squares and defines the link's control points to change the link's start (Link.StartPos property) and end (Link.EndPos property) position. The exControlPoint points are marked black circles and defines the corners of the link's path. You can remove a exControlPoint points by dragging to another, so intermediate exControlPoint points are removed. You can move all control points of the link at once, if SHIFT key is pressed. The exMiddleControlPoint points are marked with gray circles, and are displayed between two exControlPoint points, to let the use add new exControlPoint points, to redefine the link's path. The [LayoutStartChanging\(exLinkControlPoint\)](#) / [LayoutEndChanging\(exLinkControlPoint\)](#) events as soon as user starts / ends changing the link's control points

The [CustomPath](#) property specifies the link's custom path, as a string of x,y proportions separated by comma. The CustomPath property contains the proportions of link's control-points, as a "x,y,x,y,x,y,...". The x, y are proportions of link's control-points relative to the start/end points of the link. The 0,0 indicates the link's start point, while 1,1 indicates the link's end point. For instance, "0.5,0,0.5,1" defines the link to go from start (0,0) to (0.5,0), then (0.5,1), and finally to the end (1,1)



# property Surface.AllowLinkObjects as AllowKeysEnum

Specifies the combination of keys that allows the user to link the objects.

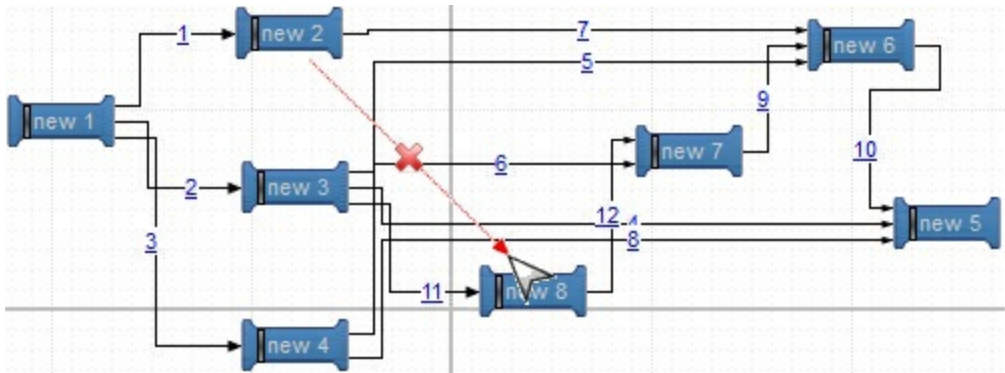
Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user can link two elements on the surface.

By default, the AllowLinkObjects property is exLeftClick + exSHIFTKey, which indicates that the user can start linking two elements once he clicks an element while keeping the SHIFT key pressed. The AllowLinkObjects property specifies the combination of keys that allows the user to link the objects. Set the AllowLinkObjects property on exDisallow to prevent adding links between elements at runtime. The [Background](#)(exLinkObjectsInvalidColor) property specifies the color to show the invalid link. The [Background](#)(exLinkObjectsValidColor) property specifies the color to show the valid link. The control fires the [LayoutStartChanging](#)(exLinkObjects) / [LayoutEndChanging](#)(exLinkObjects) event when the user creates a new link on the surface.

The order of the events when the user links two elements at runtime is:

- [LayoutStartChanging](#)(exLinkObjects), the user clicks on the surface
- [AllowLink](#), occurs to specify whether the link between two elements is possible.
- [AddLink](#), adds the new link to the Links collection
- [CreateLink](#), the user ends creating the link
- [LayoutEndChanging](#)(exLinkObjects), the user un-clicks the surface

The following screen shot shows the link when the user links elements:



# property Surface.AllowMoveDescendents as Boolean

Specifies whether all descendents of the focusing element are moved once the focusing element is moved.

Type	Description
Boolean	A Boolean expression that specifies whether all descendents elements are moved when focused element is moved.

By default, the AllowMoveDescendents property is True. Use the AllowMoveDescendents property to prevent moving all descendents ( children and outgoing elements ) when focused element is moved. The [OutgoingLinks](#) property returns a safe array of outgoing links ( links that starts from the element ). The [Children](#) property returns the collection of child elements. Use the [AllowResizeSelection](#) property to prevent resizing the entire selection when focused element is resized. Use the [AllowMoveSelection](#) property to prevent moving the entire selection when focused element is moved. Use the [ExpandLinkedElements](#) property to add expand/collapse glyphs next to elements that has outgoing links.

# property Surface.AllowMoveNeighbors as MoveNeighborsEnum

Indicates whether the neighbor elements are shifted once the selection is moved or resized, so they won't intersect the dragging objects.

Type	Description
<a href="#">MoveNeighborsEnum</a>	A MoveNeighborsEnum expression that specifies whether the neighbor elements are shifted once the selection is moved or resized, so they won't intersect the dragging objects

By default, the AllowMoveNeighbors property is exDisallowMoveNeighbors, which indicates that no neighbor elements is moved. The AllowMoveNeighbors property indicates whether the neighbor elements are shifted once the selection is moved or resized, so they won't intersect the dragging objects.

# property Surface.AllowMoveObject as AllowKeysEnum

Specifies the combination of keys that allows the user to move the objects.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user can move the element from the cursor.

By default, the AllowMoveObject property is exLeftClick, which indicates that with a simple click on the element, the user can move the element. The AllowMoveObject property specifies the keys combination so the user can move the element from the cursor. The [AllowMoveSelection](#) property indicates whether the entire selection is moved if an element in the selection is moved. Set the AllowMoveObject property on exDisallow, to prevent user to move any element in the surface. Use the [AllowResizeObject](#) property to specify whether the element can be resized at runtime. Use the [Selectable](#) property to specify whether the user can select or not an element from the surface. Since the user can not select an element, it can not move it too. The [AllowMoveSurface](#) property specifies the combination of keys that allows the user to move the surface. The control fires the [LayoutStartChanging](#)(exMoveObject) / [LayoutEndChanging](#)(exMoveObject) event when the user moves the object to a new position.

# property Surface.AllowMoveSelection as Boolean

Specifies whether the entire selection is moved once the focusing element is moved.

Type	Description
Boolean	A Boolean expression that specifies whether the entire selection is moved when focused element is moved.

By default, the AllowMoveSelection property is True. Use the AllowMoveSelection property to prevent moving the entire selection when focused element is moved. Use the [AllowResizeSelection](#) property to prevent resizing the entire selection when focused element is resized. Use the [AllowMoveDescendents](#) property to prevent moving all descendents ( children and outgoing elements ) when focused element is moved. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. The [AllowMoveObject](#) property specifies the keys combination so the user can move the element from the cursor.



## property Surface.AllowMoveSurface as AllowKeysEnum

Specifies the combination of keys that allows the user to move the surface.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user can scroll or move the surface to a new position.

By default, the AllowMoveSurface property is exLeftClick, which means you can scroll or move the surface by clicking the left mouse button and drag it to a new position. The AllowMoveSurface property specifies the combination of keys that allows the user to move the surface. The [AllowZoomSurface](#) property specifies whether the user can zoom the surface. The [AllowZoomWheelSurface](#) property specifies whether the surface is zooming when the user rotates the mouse wheel. The [AllowMoveObject](#) property specifies whether the user move the objects as soon as clicking the element. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position. The control fires the [LayoutStartChanging](#)(exSurfaceMove) / [LayoutEndChanging](#)(exSurfaceMove) event when the user moves the surface to a new position.

# property Surface.AllowResizeObject as AllowKeysEnum

Specifies the combination of keys that allows the user to resize the objects.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user can resize the element from the cursor.

By default, the AllowResizeObject property is exLeftClick, which indicates that with a simple click on the element's border, the user can resize the element. Use the AllowResizeObject property to specify whether the element can be resized at runtime. The [AllowMoveObject](#) property specifies the keys combination so the user can move the element from the cursor. The [AllowResizeSelection](#) property indicates whether the entire selection is resized if an element in the selection is resized. Set the AllowResizeObject property on exDisallow, to prevent user to resize any element in the surface. Use the [Resizable](#) / [AutoSize](#) property to prevent an element to be resized at runtime. The control fires the [LayoutStartChanging](#)(exResizeObject) / [LayoutEndChanging](#)(exResizeObject) event when the user resizes the object.

# property Surface.AllowResizeSelection as Boolean

Specifies whether the entire selection is resized once the focusing element is resize.

Type	Description
Boolean	A Boolean expression that specifies whether all elements in the selection are resized once the focused element is resized.

By default, the AllowResizeSelection property is True. Use the AllowResizeSelection property to prevent resizing the entire selection when focused element is resized. Use the [AllowMoveSelection](#) property to prevent moving the entire selection when focused element is moved. Use the [AllowMoveDescendents](#) property to prevent moving all descendents ( children and outgoing elements ) when focused element is moved. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. Use the [AllowResizeObject](#) property to specify whether the element can be resized at runtime.

# property Surface.AllowSelectNothing as Boolean

Empties the selection when the user clicks outside of the elements.

Type	Description
Boolean	A Boolean expression that specifies whether the selection is cleared when user clicks any empty part of the surface.

By default, the AllowSelectNothing property is True, which indicates that all elements are unselected when user clicks an empty part of the surface. The AllowSelectNothing property indicates whether the selection is cleared once the user clicks any empty area on the surface. Set the AllowSelectNothing property on False to prevent un-selecting elements when clicking any empty part of rge surface. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The control fires the [LayoutStartChanging](#)(exSelectNothing) / [LayoutEndChanging](#)(exSelectNothing) event when the user selects nothing ( click an empty area on the surface).

# property Surface.AllowSelectObject as AllowKeysEnum

Specifies the combination of keys that allows the user to select objects on the surface.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user can select elements

By default, the AllowSelectObject property is exLeftClick, which indicates that the user selects an element from the point when left clicking the element ( [Selectable](#) property is True ). Set the AllowSelectObject property on exDisallow to prevent selecting elements when clicking them. The AllowSelectObject property indicates the keys combination to allow user selecting new elements. The [AllowToggleSelectKey](#) property indicates the key to be used so the user can toggle a selected element. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The control fires the [LayoutStartChanging](#)(exSelectObject) / [LayoutEndChanging](#)(exSelectObject) event when the user selects the object. The [HideSel](#) property specifies whether the selected elements are highlighted or not when the control loses the focus.

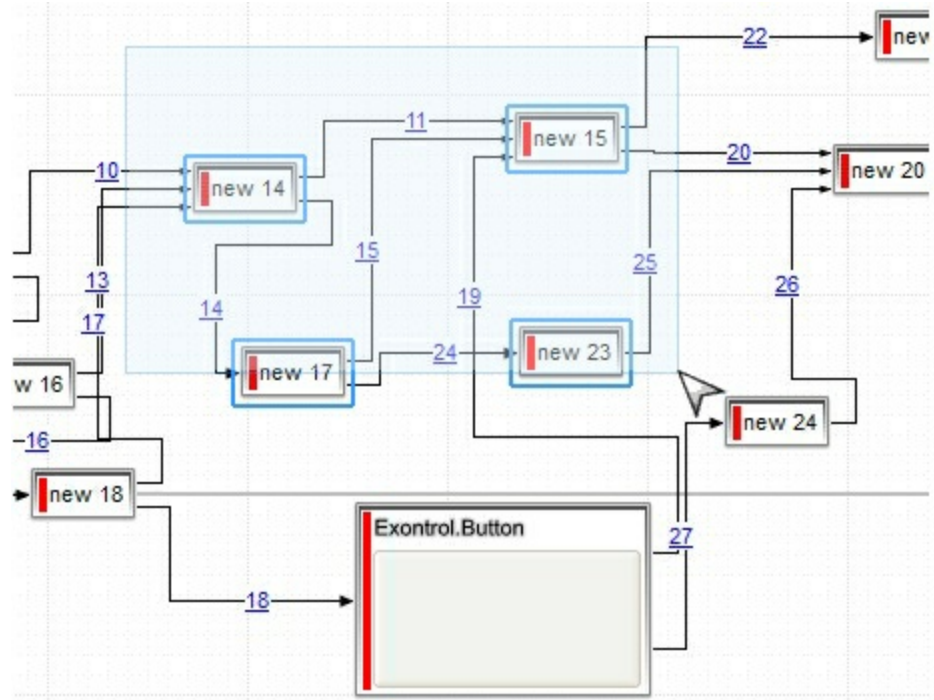
# property Surface.AllowSelectObjectRect as AllowKeysEnum

Specifies the combination of keys that allows the user to select objects on the surface, by dragging a rectangle.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user can select elements that intersects the dragging rectangle.

By default, the AllowSelectObjectRect is exLeftClick + exALTKey, which indicates that the user can start dragging a rectangle by clicking the left mouse button while keeping the ALT key, to select the elements ( [Selectable](#) property is True ) that intersect the rectangle. Set the AllowSelectObjectRect property on exDisallow to prevent selecting elements using the dragging rectangle. The AllowSelectObjectRect property specifies the keys combination so the user can select the elements from the dragging rectangle. Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowToggleSelectKey](#) property indicates the key to be used so the user can toggle a selected element. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements.

The following screen shot shows the selection rectangle:



# property Surface.AllowToggleSelectKey as AllowKeysEnum

Specifies the combination of keys to select multiple not-contiguously objects.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys to allow the user to select/unselect an element, in the surface view

By default, the AllowToggleSelectKey property is exCTRLKey, which indicates that the user can unselect/select an element by pressing the left mouse button and keeping the CTRL key down. The AllowToggleSelectKey property indicates the key to be used so the user can toggle a selected element. Set the AllowToggleSelectKey property on exDisallow to prevent selecting elements when clicking them ( while keeping the CTRL key ). The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements.



# property Surface.AllowUndoRedo as Boolean

Enables or disables the Undo/Redo feature.

Type	Description
Boolean	A boolean expression that specifies whether the control supports Undo/Redo feature

By default, the AllowUndoRedo property is false, which indicates that the Undo/Redo feature is disabled. The Undo and Redo features let you remove or repeat single or multiple actions, but all actions must be undone or redone in the order you did or undid them; you can't skip actions. For example, if you added three elements and then decide you want to undo the first change you made, you must undo all three changes. To undo an action you need to press Ctrl+Z, while for to redo something you've undone, press Ctrl+Y. The [CanUndo](#) property retrieves a value that indicates whether the control may perform the last Undo operation. The [CanRedo](#) property retrieves a value that specifies whether the control can execute the next operation in the control's Redo queue. Call the [Undo](#) method to Undo the last control operation. The [Redo](#) redoes the next action in the control's redo queue. The [UndoRedoQueueLength](#) property gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue, or in other words how many operations the control's Undo/Redo manager may store.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

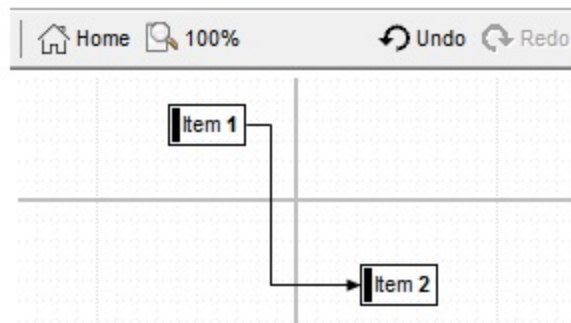
Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- **"EndBlock"**, specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The

[RedoListAction](#) property lists the Redo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked.

The control's toolbar displays the Undo/Redo commands if the [ToolBarFormat](#) property includes **103**, Undo (undoes the last control operation, enabled only if the Undo operation is possible) and **104**, Redo (redoes the next action in the control's redo queue, enabled only if the Undo operation is possible) identifiers as in the following screen shot:



The Undo/Redo toolbar-commands are automatically enabled or disabled

# property Surface.AllowZoomSurface as AllowKeysEnum

Specifies the combination of keys that allows the user to magnify or shrink the surface.

Type	Description
<a href="#">AllowKeysEnum</a>	An AllowKeysEnum expression that specifies the keys combination so the user zooms the surface

By default, the AllowZoomSurface property is exMiddleClick, which indicates that clicking the middle mouse button zooms the surface. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface. The [Zoom](#) property specifies the current zooming factor of the surface. The [ZoomMin/ZoomMax](#) property specifies the range of the surface's zooming. The [AllowZoomWheelSurface](#) property specifies whether the user can zoom the surface by rotating the mouse wheel.

The following samples shows how you can prevent zooming the surface:

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

```
With Surface1
    .AllowZoomSurface = 0
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB6

```
With Surface1
    .AllowZoomSurface = exDisallow
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB.NET

```
With Exsurface1
    .AllowZoomSurface = exontrol.EXSURFACELib.AllowKeysEnum.exDisallow
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB.NET for /COM

With AxSurface1

.AllowZoomSurface = EXSURFACELib.AllowKeysEnum.exDisallow

.AllowZoomWheelSurface = False

.ToolBarFormat = "-1,100"

End With

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control  
    Library'  
  
    #import <ExSurface.dll>  
    using namespace EXSURFACELib;  
*/  
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-  
>GetControlUnknown();  
spSurface1->PutAllowZoomSurface(EXSURFACELib::exDisallow);  
spSurface1->PutAllowZoomWheelSurface(VARIANT_FALSE);  
spSurface1->PutToolBarFormat(L"-1,100");
```

## C++ Builder

```
Surface1->AllowZoomSurface = Exsurfacelib_tlb::AllowKeysEnum::exDisallow;  
Surface1->AllowZoomWheelSurface = false;  
Surface1->ToolBarFormat = L"-1,100";
```

## C#

```
exsurface1.AllowZoomSurface = excontrol.EXSURFACELib.AllowKeysEnum.exDisallow;  
exsurface1.AllowZoomWheelSurface = false;  
exsurface1.ToolBarFormat = "-1,100";
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
  Surface1.AllowZoomSurface = 0;
  Surface1.AllowZoomWheelSurface = false;
  Surface1.ToolBarFormat = "-1,100";
</SCRIPT>
```

## C# for /COM

```
axSurface1.AllowZoomSurface = EXSURFACELib.AllowKeysEnum.exDisallow;
axSurface1.AllowZoomWheelSurface = false;
axSurface1.ToolBarFormat = "-1,100";
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    ;

    super();

    exsurface1.AllowZoomSurface(0/*exDisallow*/);
    exsurface1.AllowZoomWheelSurface(false);
    exsurface1.ToolBarFormat("-1,100");
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do
begin
  AllowZoomSurface := EXSURFACELib.AllowKeysEnum.exDisallow;
  AllowZoomWheelSurface := False;
  ToolBarFormat := '-1,100';
end
```

## Delphi (standard)

```
with Surface1 do
begin
    AllowZoomSurface := EXSURFACELib_TLB.exDisallow;
    AllowZoomWheelSurface := False;
    ToolBarFormat := '-1,100';
end
```

## VFP

```
with thisform.Surface1
    .AllowZoomSurface = 0
    .AllowZoomWheelSurface = .F.
    .ToolBarFormat = "-1,100"
endwith
```

## dBASE Plus

```
local oSurface

oSurface = form.Activex1.nativeObject
oSurface.AllowZoomSurface = 0
oSurface.AllowZoomWheelSurface = false
oSurface.ToolBarFormat = "-1,100"
```

## XBasic (Alpha Five)

```
Dim oSurface as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
oSurface.AllowZoomSurface = 0
oSurface.AllowZoomWheelSurface = .f.
oSurface.ToolBarFormat = "-1,100"
```

## Visual Objects

```
oDCOCX_Exontrol1:AllowZoomSurface := exDisallow  
oDCOCX_Exontrol1:AllowZoomWheelSurface := false  
oDCOCX_Exontrol1:ToolBarFormat := "-1,100"
```

## PowerBuilder

```
OleObject oSurface
```

```
oSurface = ole_1.Object  
oSurface.AllowZoomSurface = 0  
oSurface.AllowZoomWheelSurface = false  
oSurface.ToolBarFormat = "-1,100"
```

## Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Set ComAllowZoomSurface to OLEexDisallow  
    Set ComAllowZoomWheelSurface to False  
    Set ComToolBarFormat to "-1,100"  
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
    LOCAL oForm  
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
    LOCAL oSurface  
  
    oForm := XbpDialog():new( AppDesktop() )  
    oForm:drawingArea:clipChildren := .T.  
    oForm:create( „{100,100}, {640,480},„ .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(, {10,60},{610,370} )

oSurface:AllowZoomSurface := 0/*exDisallow*/
oSurface:AllowZoomWheelSurface := .F.
oSurface:ToolBarFormat := "-1,100"

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```



# property Surface.AllowZoomWheelSurface as Boolean

Enables or disables zooming the control using the mouse wheel.

Type	Description
Boolean	A Boolean expression that specifies whether the user can zoom the surface by rotating the mouse wheel.

By default, the AllowZoomWheelSurface property is True. The AllowZoomWheelSurface property specifies whether the user can zoom the surface by rotating the mouse wheel. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface. The [Zoom](#) property specifies the current zooming factor of the surface. The [ZoomMin/ZoomMax](#) property specifies the range of the surface's zooming.

The following samples shows how you can prevent zooming the surface:

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

```
With Surface1
    .AllowZoomSurface = 0
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB6

```
With Surface1
    .AllowZoomSurface = exDisallow
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB.NET

```
With Exsurface1
    .AllowZoomSurface = exontrol.EXSURFACELib.AllowKeysEnum.exDisallow
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB.NET for /COM

With AxSurface1

.AllowZoomSurface = EXSURFACELib.AllowKeysEnum.exDisallow

.AllowZoomWheelSurface = False

.ToolBarFormat = "-1,100"

End With

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control  
    Library'  
  
    #import <ExSurface.dll>  
    using namespace EXSURFACELib;  
*/  
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-  
>GetControlUnknown();  
spSurface1->PutAllowZoomSurface(EXSURFACELib::exDisallow);  
spSurface1->PutAllowZoomWheelSurface(VARIANT_FALSE);  
spSurface1->PutToolBarFormat(L"-1,100");
```

## C++ Builder

```
Surface1->AllowZoomSurface = Exsurfacelib_tlb::AllowKeysEnum::exDisallow;  
Surface1->AllowZoomWheelSurface = false;  
Surface1->ToolBarFormat = L"-1,100";
```

## C#

```
exsurface1.AllowZoomSurface = excontrol.EXSURFACELib.AllowKeysEnum.exDisallow;  
exsurface1.AllowZoomWheelSurface = false;  
exsurface1.ToolBarFormat = "-1,100";
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"
id="Surface1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
  Surface1.AllowZoomSurface = 0;
  Surface1.AllowZoomWheelSurface = false;
  Surface1.ToolBarFormat = "-1,100";
</SCRIPT>
```

## C# for /COM

```
axSurface1.AllowZoomSurface = EXSURFACELib.AllowKeysEnum.exDisallow;
axSurface1.AllowZoomWheelSurface = false;
axSurface1.ToolBarFormat = "-1,100";
```

## X++ (Dynamics Ax 2009)

```
public void init()
{
    ;

    super();

    exsurface1.AllowZoomSurface(0/*exDisallow*/);
    exsurface1.AllowZoomWheelSurface(false);
    exsurface1.ToolBarFormat("-1,100");
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do
begin
  AllowZoomSurface := EXSURFACELib.AllowKeysEnum.exDisallow;
  AllowZoomWheelSurface := False;
  ToolBarFormat := '-1,100';
end
```

## Delphi (standard)

```
with Surface1 do
begin
    AllowZoomSurface := EXSURFACELib_TLB.exDisallow;
    AllowZoomWheelSurface := False;
    ToolBarFormat := '-1,100';
end
```

## VFP

```
with thisform.Surface1
    .AllowZoomSurface = 0
    .AllowZoomWheelSurface = .F.
    .ToolBarFormat = "-1,100"
endwith
```

## dBASE Plus

```
local oSurface

oSurface = form.Activex1.nativeObject
oSurface.AllowZoomSurface = 0
oSurface.AllowZoomWheelSurface = false
oSurface.ToolBarFormat = "-1,100"
```

## XBasic (Alpha Five)

```
Dim oSurface as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
oSurface.AllowZoomSurface = 0
oSurface.AllowZoomWheelSurface = .f.
oSurface.ToolBarFormat = "-1,100"
```

## Visual Objects

```
oDCOCX_Exontrol1:AllowZoomSurface := exDisallow  
oDCOCX_Exontrol1:AllowZoomWheelSurface := false  
oDCOCX_Exontrol1:ToolBarFormat := "-1,100"
```

## PowerBuilder

```
OleObject oSurface
```

```
oSurface = ole_1.Object  
oSurface.AllowZoomSurface = 0  
oSurface.AllowZoomWheelSurface = false  
oSurface.ToolBarFormat = "-1,100"
```

## Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Set ComAllowZoomSurface to OLEexDisallow  
    Set ComAllowZoomWheelSurface to False  
    Set ComToolBarFormat to "-1,100"  
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
    LOCAL oForm  
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
    LOCAL oSurface  
  
    oForm := XbpDialog():new( AppDesktop() )  
    oForm:drawingArea:clipChildren := .T.  
    oForm:create( „{100,100}, {640,480},„ .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(, {10,60},{610,370} )

oSurface:AllowZoomSurface := 0/*exDisallow*/
oSurface:AllowZoomWheelSurface := .F.
oSurface:ToolBarFormat := "-1,100"

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

# property Surface.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 element'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.

The following VB sample displays ( as tooltip ) the identifier of the anchor element from the cursor:

```
Private Sub Surface1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With Surface1
        .ShowToolTip .AnchorFromPoint(-1, -1)
    End With
End Sub
```

The following VB.NET sample displays ( as tooltip ) the identifier of the anchor element from the cursor:

```
Private Sub AxSurface1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As AxEXSURFACELib._ISurfaceEvents_MouseMoveEvent) Handles AxSurface1.MouseMoveEvent
    With AxSurface1
        .ShowToolTip(.get_AnchorFromPoint(-1, -1))
    End With
End Sub
```

The following C# sample displays ( as tooltip ) the identifier of the anchor element from the cursor:

```
private void axSurface1_MouseMoveEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_MouseMoveEvent e)
{
    axSurface1.ShowToolTip(axSurface1.get_AnchorFromPoint(-1, -1));
}
```

The following C++ sample displays ( as tooltip ) the identifier of the anchor element from the cursor:

```
void OnMouseMoveSurface1(short Button, short Shift, long X, long Y)
{
    COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
    m_surface.ShowToolTip( m_surface.GetAnchorFromPoint( -1, -1 ), vtEmpty, vtEmpty,
vtEmpty );
}
```

The following VFP sample displays ( as tooltip ) the identifier of the anchor element from the cursor:

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

with thisform
    With .Surface1
        .ShowToolTip(.AnchorFromPoint(-1, -1))
    EndWith
endwith
```

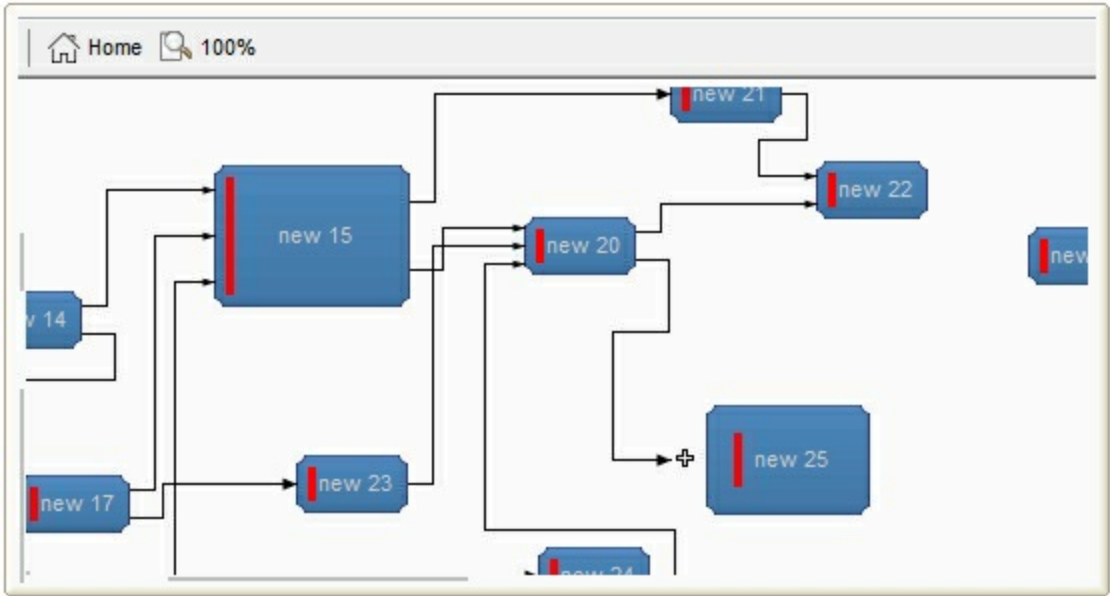


# property Surface.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
<a href="#">AppearanceEnum</a>	<p>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. <b><i>The Client object in the skin, defines the client area of the control. The list/hierarchy/chart, scrollbars are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The <a href="#">normal.ebn</a> file contains such of objects. Use the <a href="#">eXButton's Skin builder</a> to view or change this file</i></b></p>

Use the Appearance property to specify the control's border. Use the [Add](#) method to add new skins to the control. Use the [BackColor](#) property to specify the control's background color. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. You can use the [VisualDesign](#) to change the control's visual appearance at design time.



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

With Surface1

.BeginUpdate

.VisualAppearance.Add &H16, "c:\temp\normal.ebn"

.Appearance = &H16000000

.BackColor = RGB(250, 250, 250)

.EndUpdate

End With

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

With AxSurface1

.BeginUpdate()

.VisualAppearance.Add(&H16, "c:\temp\normal.ebn")

.Appearance = &H16000000

.BackColor = Color.FromArgb(250, 250, 250)

.EndUpdate()

End With

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

axSurface1.BeginUpdate();

axSurface1.VisualAppearance.Add(0x16, "c:\\temp\\normal.ebn");

axSurface1.Appearance = (EXSURFACELib.AppearanceEnum)0x16000000;

axSurface1.BackColor = Color.FromArgb(250, 250, 250);

axSurface1.EndUpdate();

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

m\_g2antt.BeginUpdate();

m\_g2antt.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\normal.ebn" ) );

m\_g2antt.SetAppearance( 0x16000000 );

m\_g2antt.SetBackColor( RGB(250,250,250) );

m\_g2antt.EndUpdate();

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

with thisform.Surface1

.BeginUpdate

.VisualAppearance.Add(0x16, "c:\temp\normal.ebn")

```
.Appearance = 0x16000000
```

```
.BackColor = RGB(250, 250, 250)
```

```
.EndUpdate
```

```
endwith
```

## method Surface.Arrange ([ID as Variant])

Arranges the elements, starting from giving element, based on the links.

Type	Description
ID as Variant	A long expression that specifies the identifier of the element where the arrangement should start, or missing to arrange ell elements of the control.

The `Arrange( ID )` method arranges the elements starting from element with the specified ID. If missing, all linked-elements are arranged. The Auto-Arrange feature arranges automatically horizontally or vertically the elements on the surface based on their relations, so they won't intersect one with another as much as possible. The [Add](#) ( of Elements collection ) method adds a new element on the surface, while [Add](#) ( of Links collection ) adds a new link between two elements. The [DefArrange](#) property defines options to perform the arrangement of the elements.

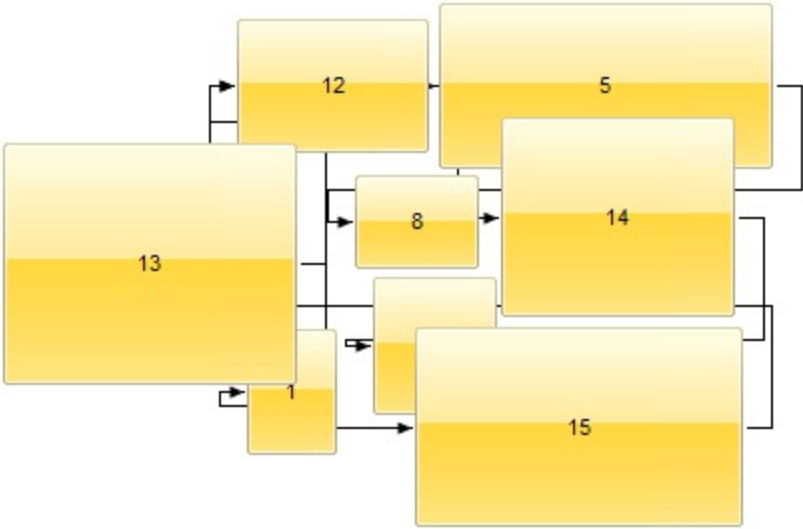
The Arrange method changes the position of the elements determined by:

- [X](#) property, specifies the element's x-position.
- [Y](#) property, specifies the element's y-position.

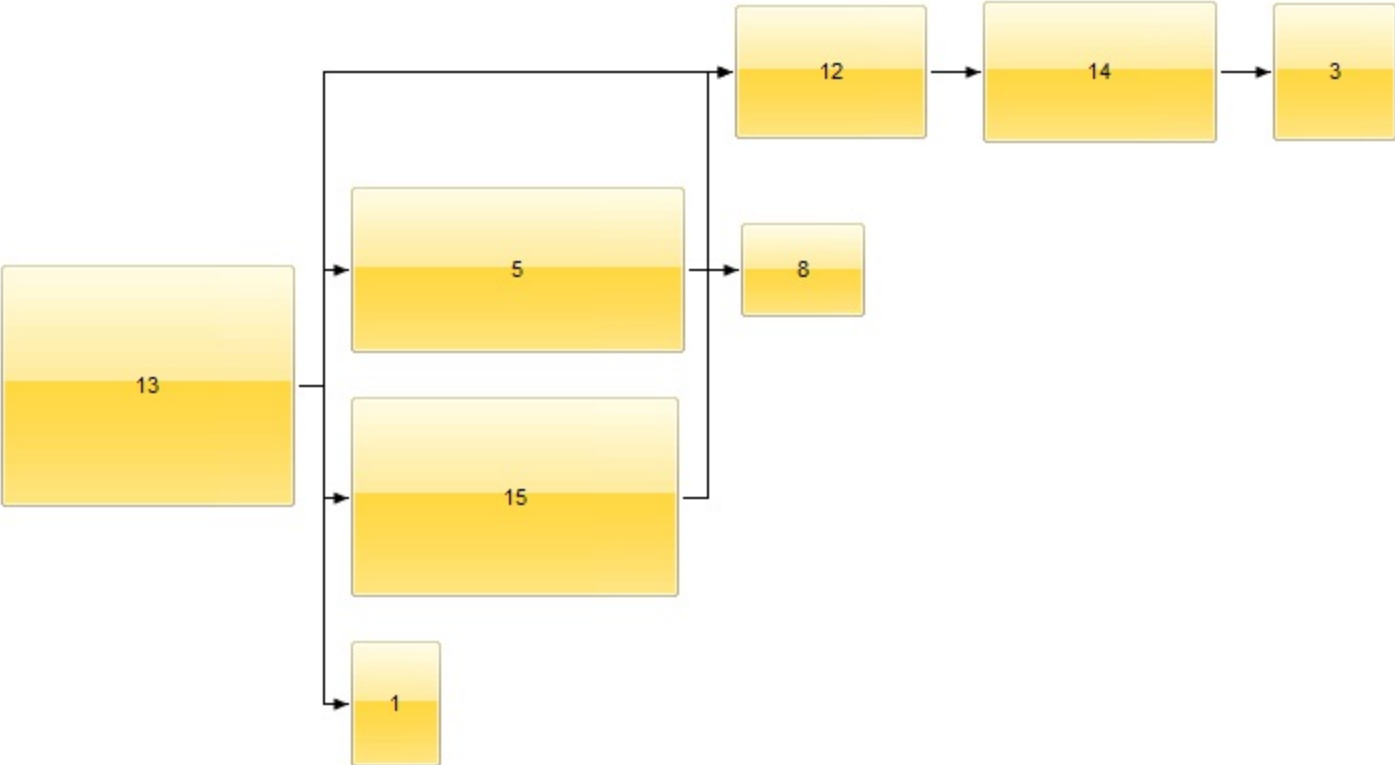
The DefArrange property can:

- arrange elements horizontally or vertically
- increases or decrease the distance between arranged elements.
- align the elements based on the incoming outgoing elements.

The following screen shot shows the elements before calling Arrange method:



The following screen shot shows the elements after calling Arrange method:



# method Surface.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 Surface1_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 Surface.AxisColor as Color

Indicates the color to show the axis on the surface.

Type	Description
Color	A Color expression that indicates the color to show the axis lines.

Use the AxisColor property to specify the color to show the axis lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.



# property Surface.AxisStyle as LinesStyleEnum

Specifies the style to display the axis lines.

Type	Description
<a href="#">LinesStyleEnum</a>	A LinesStyleEnum expression that specifies the style to show the axis lines.

By default, the AxisStyle property is exLinesSolid + exLinesThick. Use the AxisStyle property to hide the axis lines or to display with a different style. Set the AxisStyle property on exNoLines to show no axis lines. Use the [AxisColor](#) property to specify the color to show the axis lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

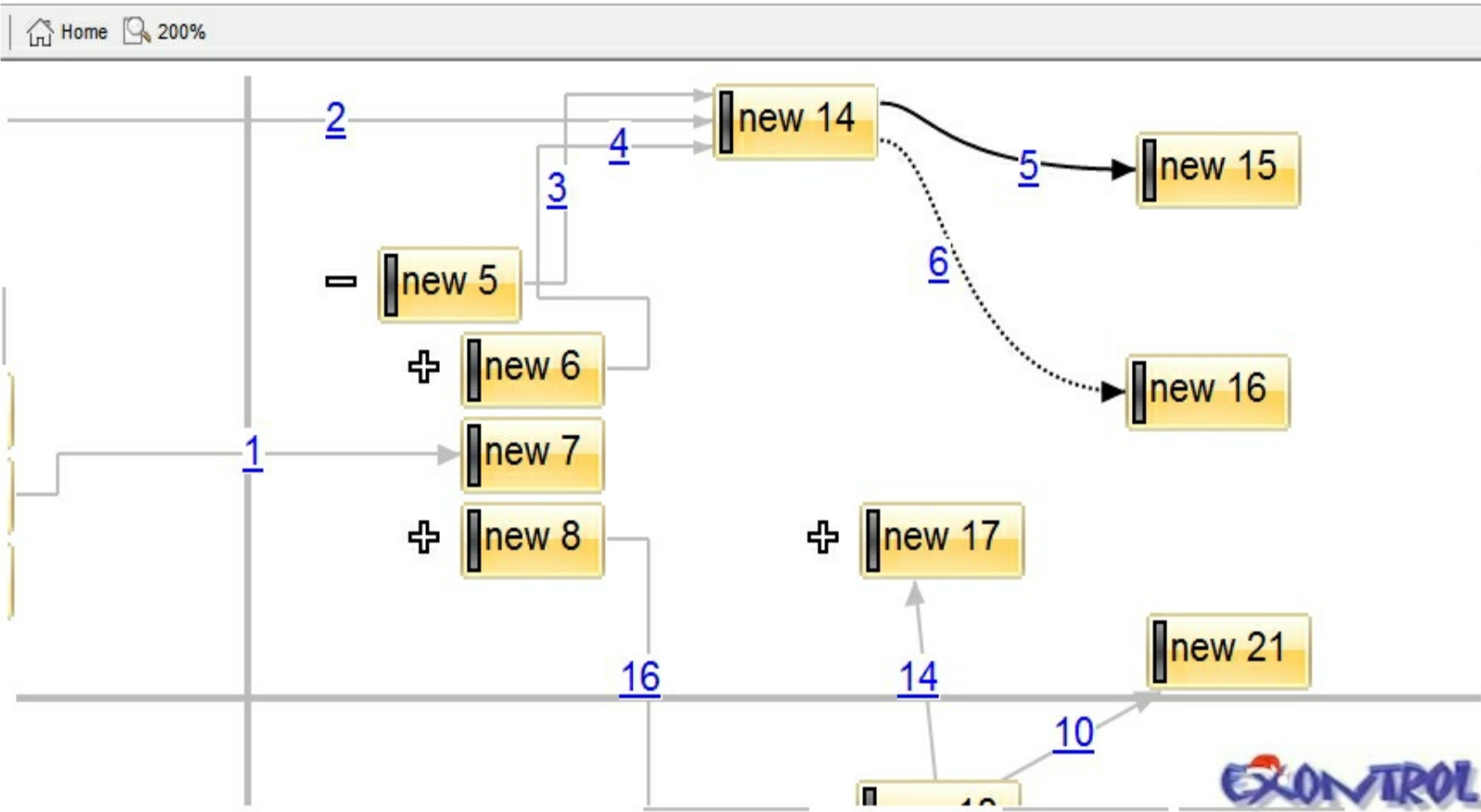
# property Surface.BackgroundColor as Color

Specifies the control's background color.

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

Use the BackColor property to specify a solid color on the control's background. The [Picture](#) property to assign your logo on the control's background. The control uses the [PictureDisplay](#) property to determine how the picture is displayed on the control's background. The [Background](#)(exElementBorderColor) or [Background](#)(exElementBackColor) property specifies the default element's border or background color. The [ForeColor](#) property specifies the color to show the captions on the elements.

The following screen shot shows a logo on the control's background:



# property Surface.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(exElementBackColor) property to specify a solid color to be shown on the element's background.

The following samples remove the border for all elements:

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

```
With Surface1
    .Background(88) = -1
    .Elements.Add "new element"
End With
```

## VB6

```
With Surface1
    .Background(exElementBorderColor) = -1
    .Elements.Add "new element"
End With
```

## VB.NET

With Exsurface1

```
.set_Background32(exontrol.EXSURFACELib.BackgroundPartEnum.exElementBorderColor)  
  
    .Elements.Add("new element")  
End With
```

## VB.NET for /COM

With AxSurface1

```
    .set_Background(EXSURFACELib.BackgroundPartEnum.exElementBorderColor,-1)  
    .Elements.Add("new element")  
End With
```

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control  
    Library'  
  
    #import <ExSurface.dll>  
    using namespace EXSURFACELib;  
*/  
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-  
>GetControlUnknown();  
spSurface1->PutBackground(EXSURFACELib::exElementBorderColor,-1);  
spSurface1->GetElements()->Add("new element",vtMissing,vtMissing);
```

## C++ Builder

```
Surface1->  
> Background[Exsurfacelib_tlb::BackgroundPartEnum::exElementBorderColor] = -1;  
Surface1->Elements->Add(TVariant("new element"),TNoParam(),TNoParam());
```

## C#

```
exsurface1.set_Background32(exontrol.EXSURFACELib.BackgroundPartEnum.exElement  
  
exsurface1.Elements.Add("new element",null,null);
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"  
id="Surface1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
    Surface1.Background(88) = -1;  
    Surface1.Elements.Add("new element",null,null);  
</SCRIPT>
```

## C# for /COM

```
axSurface1.set_Background(EXSURFACELib.BackgroundPartEnum.exElementBorderCo  
  
axSurface1.Elements.Add("new element",null,null);
```

## X++ (Dynamics Ax 2009)

```
public void init()  
{  
    ;  
  
    super();  
  
    exsurface1.Background(88/*exElementBorderColor*/,-1);  
    exsurface1.Elements().Add("new element");  
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do  
begin
```

```
set_Background(EXSURFACELib.BackgroundPartEnum.exElementBorderColor,$ffffff);

    Elements.Add('new element',Nil,Nil);
end
```

## Delphi (standard)

```
with Surface1 do
begin
    Background[EXSURFACELib_TLB.exElementBorderColor] := $ffffff;
    Elements.Add('new element',Null,Null);
end
```

## VFP

```
with thisform.Surface1
.Object.Background(88) = -1
.Elements.Add("new element")
endwith
```

## dBASE Plus

```
local oSurface

oSurface = form.Activex1.nativeObject
oSurface.Template = [Background(88) = -1] // oSurface.Background(88) = -1
oSurface.Elements.Add("new element")
```

## XBasic (Alpha Five)

```
Dim oSurface as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
oSurface.Template = "Background(88) = -1" ' oSurface.Background(88) = -1
oSurface.Elements.Add("new element")
```

## Visual Objects

```
oDCOCX_Exontrol1:[Background,exElementBorderColor] := -1
oDCOCX_Exontrol1:Elements:Add("new element",nil,nil)
```

## PowerBuilder

```
OleObject oSurface

oSurface = ole_1.Object
oSurface.Background(88,-1)
oSurface.Elements.Add("new element")
```

## Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Set ComBackground OLEexElementBorderColor to -1
    Variant voElements
    Get ComElements to voElements
    Handle hoElements
    Get Create (RefClass(cComElements)) to hoElements
    Set pvComObject of hoElements to voElements
        Get ComAdd of hoElements "new element" Nothing Nothing to Nothing
    Send Destroy to hoElements
End_Procedure
```

## XBase++

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

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oSurface
```

```

oForm := XbpDialog():new( AppDesktop() )
oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(,, {10,60},{610,370} )

oSurface:SetProperty("Background",88/*exElementBorderColor*/,-1)
oSurface:Elements():Add("new element")

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```



# method Surface.BeginUpdate ()

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

Type	Description
	This method prevents the control from painting until the <a href="#">EndUpdate</a> method is called. Use the <a href="#">Refresh</a> method to refresh the control.

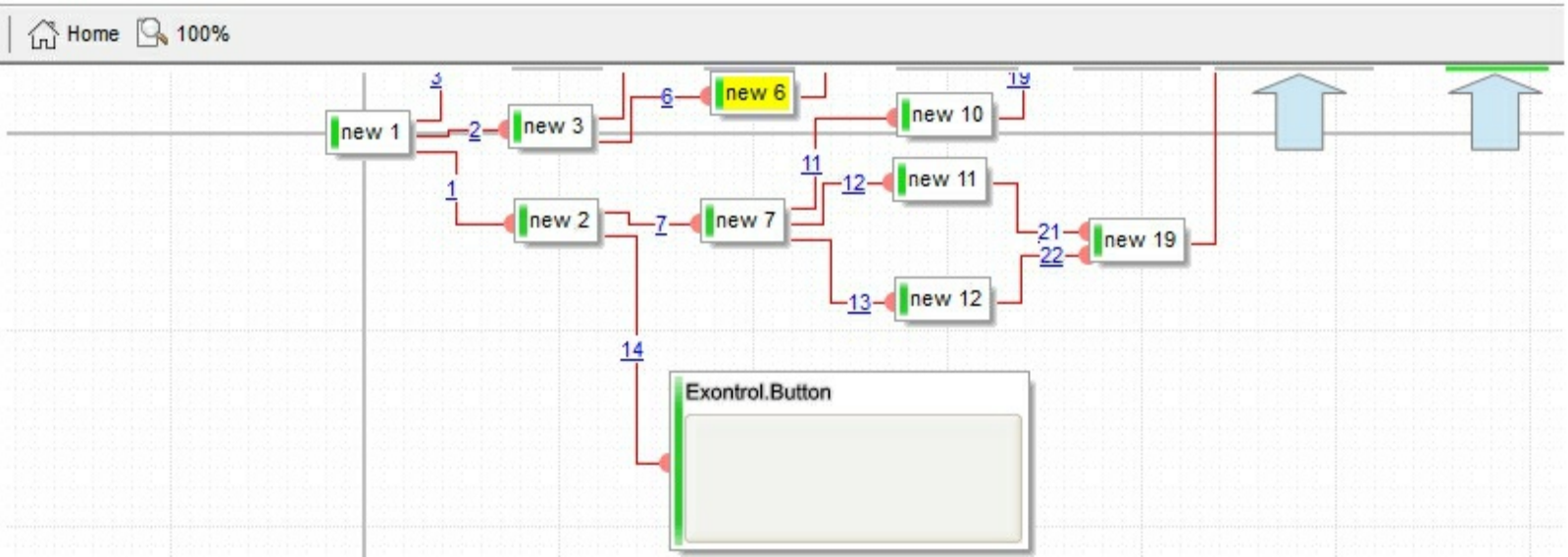
# property Surface.BorderHeight as Long

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

Type	Description
Long	A Long expression that specifies the height of the control's border where the element that does not fit the control's visible part are shown.

By default, the BorderHeight property is 2 pixels tall. The [BorderWidth](#) / BorderHeight properties specify the size on the margin where the overview elements are shown. The [OverviewColor](#) property specifies the color to show the elements when they do not fit the surface's visible area. The [OverviewColor](#) property has effect when the element is not fitting the surface's client area and it is shown on the border of the surface.

The following screen shot shows the how elements are shown when they are not visible in the surface's client area ( look on the border ) :



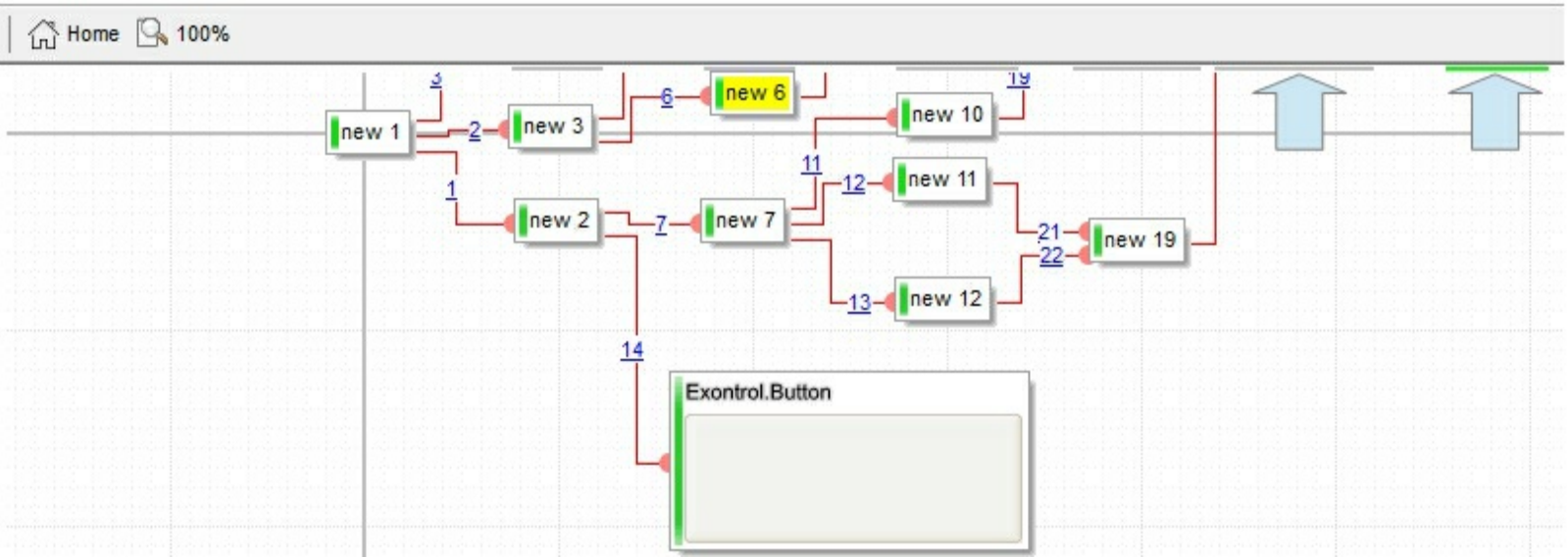
# property Surface.BorderWidth as Long

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

Type	Description
Long	A Long expression that specifies the width of the control's border where the element that does not fit the control's visible part are shown.

By default, the BorderWidth property is 2 pixels wide. The BorderWidth / [BorderHeight](#) properties specify the size on the margin where the overview elements are shown. The [OverviewColor](#) property specifies the color to show the elements when they do not fit the surface's visible area. The [OverviewColor](#) property has effect when the element is not fitting the surface's client area and it is shown on the border of the surface.

The following screen shot shows the how elements are shown when they are not visible in the surface's client area ( look on the border ) :



## method **Surface.CancelLayoutChanging ()**

Cancels the current layout changing operation.

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

During the [LayoutStartChanging](#) event, you can call the CancelLayoutChanging method to cancel the specified operation. Calling the CancelLayoutChanging method anywhere else, has no effect.

The operations being signaled by the [LayoutStartChanging](#) / [LayoutEndChanging](#) events are:

- **exSurfaceMove**, the user scrolls or moves the surface. The [AllowMoveSurface](#) property specifies the keys combination to allow user to move / scroll the surface.
- **exSurfaceZoom**, the user magnifies or shrinks the surface. The [AllowZoomSurface](#) property specifies the keys combination to allow user to zoom the surface.
- **exSurfaceHome**, the user clicks the Home button on the control's toolbar, so the surface is restored to original position. The [Home](#) method has the same effect.
- **exResizeObject**, the user resizes the object. The [AllowResizeObject](#) property specifies the keys combination to allow user to resize the object.
- **exMoveObject**, the user moves the object. The [AllowMoveObject](#) property specifies the keys combination to allow user to move the object.
- **exSelectObject**, the user clicks the object to get it selected. The [AllowSelectObject](#) property specifies the keys combination to allow user to select the object.
- **exSelectNothing**, the user clicks an empty zone of the surface. The [AllowSelectNothing](#) property specifies the keys combination to allow user to select nothing on the surface.
- **exCreateObject**, the user creates an element on the surface. The [AllowCreateObject](#) property specifies the keys combination to allow user to create elements on the surface.
- **exEditObject**, the user edits the element's caption.
- **exLinkObjects**, the user creates an element on the surface. The [AllowLinkObjects](#) property specifies the keys combination to allow user to link elements on the surface.

# property Surface.CanRedo as Boolean

Retrieves a value that indicates whether the surface can perform a Redo operation.

Type	Description
Boolean	A boolean expression that specifies whether the control can perform a Redo operation

The CanRedo method indicates whether the control can perform a Redo operation. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [Redo](#) redoes the next action in the control's redo queue. The [Undo](#) method undoes the last control operation. The [UndoRedoQueueLength](#) property gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue, or in other words how many operations the control's Undo/Redo manager may store.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins
- **"EndBlock"**, specifies that a block of operations ends

The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked.

# property Surface.CanUndo as Boolean

Retrieves a value that indicates whether the surface can perform an Undo operation.

Type	Description
Boolean	A boolean expression that specifies whether the control can perform an Undo operation

The CanUndo method indicates whether the control can perform an Undo operation. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [Undo](#) method undoes the last control operation. The [Redo](#) redoes the next action in the control's redo queue. The [UndoRedoQueueLength](#) property gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue, or in other words how many operations the control's Undo/Redo manager may store.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins
- **"EndBlock"**, specifies that a block of operations ends

The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked.

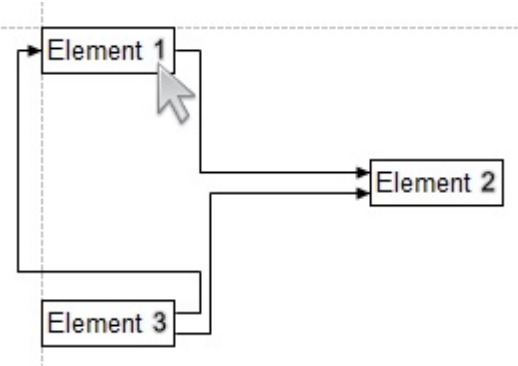
# property Surface.Coord as CoordEnum

Specifies the type of coordinates the elements of the surface display in.

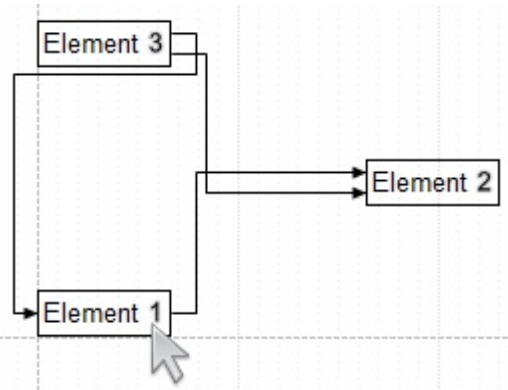
Type	Description
<a href="#">CoordEnum</a>	A CoordEnum expression that specifies the type of coordinates the elements of the surface display in.

By default, the Coord property is exDefCoord, which indicates that the positive coordinates always show bottom-right to the origin. The Coord property can be used to allow the surface to display positive coordinates only.

The following screen shot shows the surface using default coordinates:



The following screen shot shows the surface using Cartesian coordinates:



# property Surface.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>• *.bmp *.dib *.rle, saves the control's content in <b>BMP</b> format.</li><li>• *.jpg *.jpe *.jpeg *.jfif, saves the control's content in <b>JPEG</b> format.</li><li>• *.gif, , saves the control's content in <b>GIF</b> format.</li><li>• *.tif *.tiff, saves the control's content in <b>TIFF</b> format.</li><li>• *.png, saves the control's content in <b>PNG</b> format.</li><li>• *.pdf, 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>• *.emf 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, 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, including no scroll bars.

- 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 file:

```
If (Surface1.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 Surface1.CopyTo("")  
    Debug.Print i  
Next
```

# property Surface.DefArrange(Option as DefArrangeEnum) as Variant

Retrieves or sets an option for Arrange method.

Type	Description
Option as <a href="#">DefArrangeEnum</a>	A <a href="#">DefArrangeEnum</a> expression that specifies the property of the Arrange method to be accessed
Variant	A VARIANT expression that specifies the value of the giving property.

The DefArrange property retrieves or sets an option for [Arrange](#) method. The Arrange method arranges the elements, starting from giving element, based on the links. Changing any [DefArrangeEnum](#) properties has effect at the next Arrange call only.

For instance, you can use the DefArrange property to:

- arrange elements horizontally or vertically
- increases or decrease the distance between arranged elements.
- align the elements based on the incoming outgoing elements.

# property Surface.DrawPartsOrder as String

Defines the order of the parts the elements display

Type	Description
String	A string expression that specifies the order to draw the parts of the elements. The list is separated by comma, no spaces are accepted

By default, the DrawPartsOrder property is "extracaption,caption,extrapicture,picture,check,client". The DrawPartsOrder property supports the following parts (separated by comma): "extracaption", "caption", "extrapicture", "picture", "check" and "client". You can use the DrawPartsOrder property to make the caption to be displayed over the picture, by using "extrapicture,picture,check,extracaption,caption,client". If a part is missing from the DrawPartsOrder property, it is not shown in the element. For instance, if "check" is missing the element shows no check-box, even the element's ShowCheckBox property is True. The [CaptionAlign](#), [PicturesAlign](#) property aligns the caption/pictures The [ElementFormat](#) property specifies the way the control shows the parts of the element (The DrawPartsOrder property has no effect if the ElementFormat property is set)

# property Surface.EditContextMenuItems as String

Specifies the control's context menu, while editing the event.

Type	Description
String	A string expression that indicates the items to be shown on the edit's context menu.

The edit's context menu is displayed if the user right clicks while editing the event. Use the EditContextMenuItems property to change the edit's context menu.

By default the EditContextMenuItems property is:

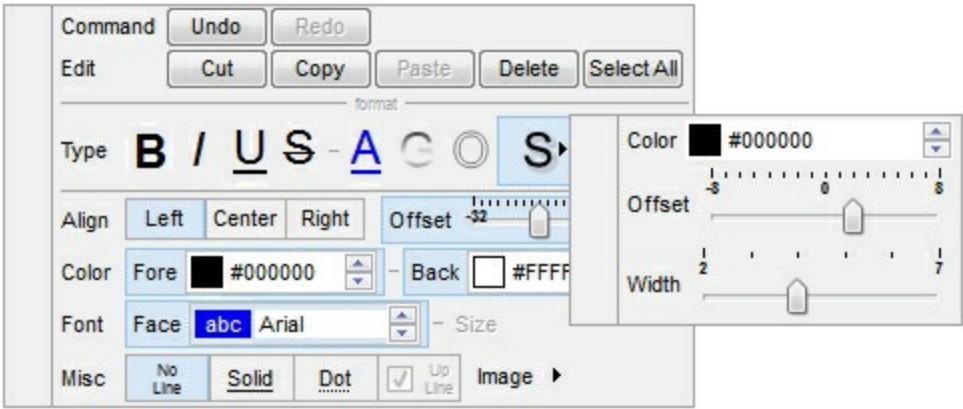
```
Command[id=57625][captionwidth=48][group=19](Undo[id=57643][align=1]
[button=-1][captionwidth=44],Redo[id=57644][align=1][button=-1]),Edit[id=57624]
[captionwidth=48][group=19](Cut[id=57635][align=1][button=-1],Copy[id=57634]
[align=1][button=-1],Paste[id=57637][align=1][button=-1],Delete[id=57632][align=1]
[button=-1],Select All[id=57642][align=1][button=-1]),format[sep][id=57623]
[height=13],Type[id=57622][show=1][captionwidth=28][group=19](B[id=57648]
[typ=1][align=1][show=1],I[id=57649][typ=1][align=1][show=1],U[id=57650]
[typ=1][align=1][show=1],S[id=57651][typ=1][align=1][show=1],[sep]
[id=57621],A[id=57760][typ=1][align=1][spchk=-1][show=1](ID[id=57761]
[edittype=1][editwidth=-172],Options[id=57762][edittype=1]
[editwidth=-72]),G[id=57715][typ=1][align=1][spchk=-1][show=1]
(Color[id=57717][edittype=518][border=0][editwidth=-72],Mode[id=57724]
[group=17](H[id=57725][typ=2][align=1][chk=1][show=1][grp=2],V[id=57726]
[typ=2][align=1][show=1][grp=2],FD[id=57727][typ=2][align=1][show=1]
[grp=2],BD[id=57728][typ=2][align=1][show=1][grp=2]),Blend Triangular
Shape[id=57729][typ=1][show=-1]),O[id=57730][typ=1][align=1][spchk=-1]
[show=1](Color[id=57732][edittype=518][border=0]
[editwidth=-96],Width[id=57739][edittype=3][border=0][min=1][max=4][freq=1]
[editwidth=-72]),S[id=57743][typ=1][align=1][spchk=-1][show=1]
(Color[id=57745][edittype=518][border=0][editwidth=-72],Offset[id=57752]
[edittype=3][border=0][min=-8][max=+8][freq=1]
[editwidth=-128],Width[id=57756][edittype=3][border=0][min=2][max=+7]
[freq=1][editwidth=-128])),[sep][id=57620][height=4],Align[id=57619][show=1]
[captionwidth=24][height=26][group=19](id=57618)[group=19]
(Offset[id=57709][typ=1][chk][show=1][showdis][border=0][min=-32][max=+32]
[freq=4][editwidth=-96][height=24])),Color[id=57618][captionwidth=28]
```

```

[height=26][group=3](Fore[id=57685][typ=1][show=1][showdis][editwidth=-96]
[height=24],[sep][id=57617],Back[id=57686][typ=1][show=1][showdis]
[editwidth=-96][height=24]),Font[id=57617][captionwidth=28][height=26]
[group=3](Face[id=57701][typ=1][show=1][showdis][height=24][editwidth=-116],
[sep][id=57616],Size[id=57702][typ=1][show=1][showdis][height=24]
[editwidth=-82][min=4][max=72][freq=4]),Misc[id=57609][captionwidth=24]
[group=3](Image[id=57608](Size[id=57680][edittype=515][border=0][min=16]
[max=128][freq=16][editwidth=-128][ticklabel=value = %i ? " + value : ( value =
vmax ? " + value : ( value = vmin ? " + value : " ) )],Insert[id=57679]()))

```

By default, the control's context menu shows as following:



Let's say we want to remove all that grouping, and shows as a regular context menu ( just remove all the [group] from the EditContextMenuItems property, and you should get something like:



The EditContextMenuItems's syntax in BNF notation:

```

<EditContextMenuItems> ::= <ITEMS>
<ITEMS> ::= <ITEM>["("<ITEMS>")"][","<ITEMS>]
<ITEM> ::= <CAPTION>[<OPTIONS>]
<OPTIONS> ::= "["<OPTION>"] "["["<OPTIONS>"]"]
<OPTION> ::= <PROPERTY>["="<VALUE>]
<PROPERTY> ::= "img" | "himg" | "sep" | "id" | "typ" | "group" | "chk" | "button" | "align" |

```

"spchk" | "show" | "rad" | "dis" | "showdis" | "bld" | "itl" | "stk" | "und" | "bg" | "fg" | "editttype" | "edit" | "mask" | "border" | "editwidth" | "captionwidth" | "height" | "grp" | "tft" | "ttp" | "min" | "max" | "tick" | "freq" | "ticklabel" | "small" | "large" | "spin" | "ettp" | "float"

where the <CAPTION> is the HTML caption to be shown on the context menu item. The <VALUE> indicates the value of giving property.

- **img=<VALUE>**, where <VALUE> is an integer expression, that indicates the index of the icon being displayed for the item.
- **himg=<VALUE>**, where <VALUE> indicates the key of the picture to be displayed for the item.
- **sep**, specifies an separator item
- **id=<VALUE>**, where <VALUE> is an integer expression, that indicates the identifier of the item.
- **typ=<VALUE>**, where <VALUE> could be one of the following:
  - **0** for regular items,
  - **1** for items that display a check/box (chk)
  - **2** to display radio buttons (rad)
- **group=<VALUE>**, where <VALUE> could be a bit-or combination (+) of the following values:
  - **0** (exNoGroupPopup), No grouping is performed on the sub-menu, so the sub-items are shown to a float popup,
  - **1** (exGroupPopup), Groups and displays the sub-menu items on the current item, arranged from left to right
  - **2** (exNoGroupPopupFrame), Prevents showing the frame around each grouping item.
  - **4** (exGroupPopupCenter), Shows the grouping popup aligned to the center of the current item.
  - **8** (exGroupPopupRight), Shows the grouping popup aligned to the right of the current item.
  - **16** (exGroupPopupEqualSize), Shows the items that make the group of the same size
- **chk[=<VALUE>]**, where <VALUE> could be **0** for unchecked, or **not zero** for checked. The chk option makes the item to display a check box. If the <VALUE> is missing the item still displays an un-checked check box.
- **button=<VALUE>**, where <VALUE> could be **0** for regular or **not zero** to show the item as a button.
- **align=<VALUE>**, where <VALUE> could be one of the following:
  - **0** ( left ), to align the item's caption to the left
  - **1** ( center ), to center the item's caption
  - **2** ( right ), to align the item's caption to the right
- **spchk=<VALUE>**, where <VALUE> could be **0** for regular or **not zero** to specify whether the item's sub menu is shown only if the item is checked.

- show=<VALUE>, where <VALUE> could be **0** for regular or **not zero** to specify whether the checked item shows as selected
- rad=<VALUE>, where <VALUE> could be **0** for unchecked radio button or **not zero** to for checked radio button. Use the grp option to define the group of radio where this button should be associated, If no group of radio buttons is required, the grp could be ignored.
- dis, specifies a disabled item
- showdis=<VALUE>, where <VALUE> could be **0** for regular or **not zero** to specify whether the item shows as disabled, but it is still enabled
- bld, specifies that the item appears in bold
- itl, specifies that the item appears in italics
- stk, specifies that the item appears as strikeout
- und, specifies that the item is underlined
- bg=<VALUE>, specifies the item's background color, where <VALUE> could be a RGB expression ( RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or a long expression.
- fg=<VALUE>, specifies the item's foreground color, where <VALUE> could be a RGB expression ( RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or a long expression.
- edittype=<VALUE>, associates an edit field to the item, where <VALUE> could be one of the following values:
  - **0** ( exlItemDisableEdit ), No editor is assigned to the current item.
  - **1** ( exlItemEditText ), A text-box editor is assigned to the current item.
  - **2** ( exlItemEditMask ), A masked text-box editor is assigned to the current item.
  - **3** ( exlItemEditSlider ), A slider editor is assigned to the current item.
  - **4** ( exlItemEditProgress ), A progress editor is assigned to the current item.
  - **5** ( exlItemEditScrollBar ), A scrollbar editor is assigned to the current item.
  - **6** ( exlItemEditColor), A color editor is assigned to the current item.
  - **7** ( exlItemEditFont ), A font editor is assigned to the current item.
  - **256** (exlItemEditReadOnly), specifies that the item's editor is shown as disabled. This value could be combined with one of the values from 0 to 7, 512
  - **512** ( exlItemEditSpin ), A spin editor is assigned to the current item. This value could be combined with one of the values from 0 to 7, 256
- edit=<VALUE>, specifies the caption to be shown in the item's edit field, where <VALUE> could be any string
- mask=<VALUE>, specifies the mask to be applied on a masked editor. This option is valid for exlItemEditMask edit. Use the float option to allow masking floating point numbers. See [Masking](#) for more information about <VALUE> of the mask option. See [Masking Float](#) for more information about <VALUE> if the float option is used.
- border=<VALUE>, specifies the border to be shown on the item's edit field, where <VALUE> could be one of the following:
  - **0** ( exEditBorderNone), No border is shown.



- **-1** (exEditBorderInset), shows an inset border
- **1** (exEditBorderSingle), shows a frame border
- editwidth=<VALUE>, specifies the width to show the edit field inside the item. where <VALUE> could be a long expression. A negative value indicates that the field goes to the end of the item
- captionwidth=<VALUE>, specifies the width to show the HTML caption of the item. where <VALUE> could be a long expression. A negative value indicates that no limitation is applied to the item's caption, so no truncate caption is shown
- height=<VALUE>, specifies the height to show the item, where <VALUE> could be a positive long expression
- grp=<VALUE>, defines the radio group. It should be used when you define more groups of radio buttons. A group of radio buttons means that only one item could be checked at one time. The rad option specifies that the item displays a radio button. Use the grp option to define the group of radio where this button should be associated, If no group of radio buttons is required, the grp could be ignored. The <VALUE> could be any long expression.
- ttp=<VALUE>, defines the item's tooltip. The <VALUE> could be any HTML string expression. The item's tooltip is shown when the user hovers the item.
- min=<VALUE>, defines the minimum value of the edit field. The <VALUE> could be any long expression, and specifies the minimum value for any slider, progress, scroll, spin, or range editor.
- max=<VALUE>, defines the maximum value of the edit field. The <VALUE> could be any long expression, and specifies the maximum value for any slider, progress, scroll, spin, or range editor.
- tick=<VALUE>, defines where the ticks of the slider edit appear. This option is valid for exItemEditSlider edit. The <VALUE> could be one of the following values:
  - **0** ( exBottomRight ), The ticks are displayed on the bottom/right side.
  - **1** ( exTopLeft ), The ticks are displayed on the top/left side.
  - **2** ( exBoth ), The ticks are displayed on the both side.
  - **3** ( exNoTicks ), No ticks are displayed.
- freq=<VALUE>, indicates the ratio of ticks on the slider edit. This option is valid for exItemEditSlider edit. The <VALUE> could be a positive long expression.
- ticklabel=<VALUE>, indicates the HTML label to be displayed on slider's ticks. This option is valid for exItemEditSlider edit. See [Tick Label Expression](#) for more information about <VALUE> of the ticklabel option.
- small=<VALUE>, indicates the amount by which the edit's position changes when the user presses the arrow key ( left, right, or button ). This option is valid for exItemEditSlider, exItemEditScrollBar edit. The <VALUE> could be a positive long expression.
- large=<VALUE>, indicates the amount by which the edit's position changes when the user presses the CTRL + arrow key ( CTRL + left, CTRL + right). This option is valid for exItemEditSlider, exItemEditScrollBar edit. The <VALUE> could be a positive long expression.

- `spin=<VALUE>`, specifies the step to advance when user clicks the editor's spin.. This option is valid for `exltemEditSpin` edit. The `<VALUE>` could be a positive long expression.
- `ettp=<VALUE>`, specifies the HTML tooltip to be shown when the item's value is changed. This option is valid for `exltemEditSlider/exltemEditScrollBar` edit. The `<VALUE>` could be any string expression.
- `float=<VALUE>`, Specifies whether the mask field masks a floating point number. This option is valid for `exltemEditMask` edit. See [Masking Float](#) for more information about `<VALUE>` of mask option, if the float option is used. The `<VALUE>` could be **0** for standard masking field or **not zero** to specify that the field is masking a floating point.

## ContextMenu - Masking

For instance, the following input-mask ( ext-phone )

*!(999) 000 0000;1;;select=1,empty,overtyp e,warning=invalid character,invalid=The value you entered isn't appropriate for the input mask <b>'<%mask%>'</b> specified for this field."*

indicates the following:

- The pattern should contain 3 optional digits 999, and 7 required digits 000 0000, aligned to the right, *!*.
- The second part of the input mask indicates 1, which means that all literals are included when the user leaves the field.
- The entire field is selected when it receives the focus, *select=1*
- The field supports *empty* value, so the user can leave the field with no content
- The field enters in *overtyp e* mode, and insert-type mode is not allowed when user pressed the Insert key
- If the user enters any invalid character, a *warning* tooltip with the message "*invalid character*" is displayed.
- If the user tries to leave the field, while the field is not validated ( all 7 required digits completed ), the *invalid* tooltip is shown with the message "*The value you entered isn't appropriate for the input mask <b>'<%mask%>'</b> specified for this field.*" The `<%mask%>` is replaced with the first part of the input mask *!(999) 000 0000*

The four parts of an input mask, or the Mask property supports up to four parts, separated by a semicolon (;). For instance, `"`Time: `00:00:00;;0;overtyp e,warning=<fgcolor FF0000>invalid character,beep"`, indicates the pattern "00:00" with the prefix Time:, the masking character being the 0, instead `_`, the field enters in over-type mode, insert-type mode is not allowed, and the field beeps and displays a tooltip in red with the message invalid character when the user enters an invalid character.

Input masks are made up one mandatory part and three optional parts, and each part is

separated by a semicolon (;). If a part should use the semicolon (;) it must use the \; instead

The purpose of each part is as follows:

1. The first part (pattern) is mandatory. It includes the mask characters or string (series of characters) along with placeholders and literal data such as, parentheses, periods, and hyphens.

The following table lists the placeholder and literal characters for an input mask and explains how it controls data entry:

- **#**, a digit, +, - or space (entry not required).
- **0**, a digit (0 through 9, entry required; plus [+] and minus [-] signs not allowed).
- **9**, a digit or space (entry not required; plus and minus signs not allowed).
- **x**, a lower case hexa character, [0-9],[a-f] ( entry required )
- **X**, an upper case hexa character, [0-9],[A-F] ( entry required )
- **A**, any letter, digit (entry required).
- **a**, any letter, digit or space (entry optional).
- **L**, any letter (entry require).
- **?**, any letter or space (entry optional).
- **&**, any character or a space (entry required).
- **C**, any character or a space (entry optional).
- **>**, any letter, converted to uppercase (entry required).
- **<**, any letter, converted to lowercase (entry required).
- **\***, any characters combinations
- **{ min,max }** (Range), indicates a number range. The syntax {min,max} (Range), masks a number in the giving range. The min and max values should be positive integers. For instance the mask {0,255} masks any number between 0 and 255.
- **[...]** (Alternative), masks any characters that are contained in the [] brackets. For instance, the [abcdA-D] mask any character: a,b,c,d,A,B,C,D
- **\**, indicates the escape character
- **t'**, ( ALT + 175 ) causes the characters that follow to be converted to uppercase, until **Ť**( ALT + 174 ) is found.
- **Ť**, ( ALT + 174 ) causes the characters that follow to be converted to lowercase, until **t'**( ALT + 175 ) is found.
- **!**, causes the input mask to fill from right to left instead of from left to right.

Characters enclosed in double quotation ("" or ``) marks will be displayed literally. If this part should display/use the semicolon (;) character is should be included between double quotation ("" or ``) characters or as \; ( escape ).

2. The second part is optional and refers to the embedded mask characters and how they are stored within the field. If the second part is set to 0 ( default, `exClipModeLiteralsNone` ), all characters are stored with the data, and if it is set to 1 (`exClipModeLiteralsInclude`), the literals are stored, not including the masking/placeholder characters, if 2 (`exClipModeLiteralsExclude`), just typed characters are stored, if 3(`exClipModeLiteralsEscape`), optional, required, editable and escaped entities are included. No double quoted text is included.
3. The third part of the input mask is also optional and indicates a single character or space that is used as a placeholder. By default, the field uses the underscore (`_`). If you want to use another character, enter it in the third part of your mask. Only the first character is considered. If this part should display/use the semicolon (`;`) character is should be `\;` ( escape )
4. The forth part of the input, indicates a list of options that can be applied to input mask, separated by comma(`,`) character.

The known options for the forth part are:

- ***float***, indicates that the field is edited as a decimal number, integer. The first part of the input mask specifies the pattern to be used for grouping and decimal separators, and - if negative numbers are supported. If the first part is empty, the float is formatted as indicated by current regional settings. For instance, `"###;;float"` specifies a 2 digit number in float format. The grouping, decimal, negative and digits options are valid if the float option is present.
- ***grouping=value***, Character used to separate groups of digits to the left of the decimal. Valid only if float is present. For instance `";;;float,grouping="` indicates that no grouping is applied to the decimal number (`LOCALE_STHOUSAND`)
- ***decimal=value***, Character used for the decimal separator. Valid only if float is present. For instance `";;;float,grouping= ,decimal=,\"` indicates that the decimal number uses the space for grouping digits to the left, while for decimal separator the comma character is used (`LOCALE_SDECIMAL`)
- ***negative=value***, indicates whether the decimal number supports negative numbers. The value should be 0 or 1. 1 means negative numbers are allowed. Else 0 or missing, the negative numbers are not accepted. Valid only if float is present.
- ***digits=value***, indicates the max number of fractional digits placed after the decimal separator. Valid only if float is present. For instance, `";;;float,digits=4"` indicates a max 4 digits after decimal separator (`LOCALE_IDIGITS`)
- ***password[=value]***, displays a black circle for any shown character. For instance,

*;;;password", specifies that the field to be displayed as a password. If the value parameter is present, the first character in the value indicates the password character to be used. By default, the \* password character is used for non-TrueType fonts, else the black circle character is used. For instance, ";;;password=\*", specifies that the field to be displayed as a password, and use the \* for password character. If the value parameter is missing, the default password character is used.*

- **right**, aligns the characters to the right. For instance, "(999) 999-9999;;;right" displays and masks a telephone number aligned to the right. **readonly**, the editor is locked, user can not update the content, the caret is available, so user can copy the text, excepts the password fields.
- **inserttype**, indicates that the field enters in insert-type mode, if this is the first option found. If the forth part includes also the overtype option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "###:###;0;inserttype,overtime", indicates that the field enter in insert-type mode, and over-type mode is allowed. The "###:###;0;inserttype", indicates that the field enter in insert-type mode, and over-type mode is not allowed.
- **overtime**, indicates that the field enters in over-type mode, if this is the first option found. If the forth part includes also the inserttype option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "###:###;0;overtime,inserttype", indicates that the field enter in over-type mode, and insert-type mode is allowed. The "###:###;0;overtime", indicates that the field enter in over-type mode, and insert-type mode is not allowed.
- **nocontext**, indicates that the field provides no context menu when user right clicks the field. For instance, ";;;password,nocontext" displays a password field, where the user can not invoke the default context menu, usually when a right click occurs.
- **beep**, indicates whether a beep is played once the user enters an invalid character. For instance, "00:00;;;beep" plays a beep once the user types in invalid character, in this case any character that's not a digit.
- **warning=value**, indicates the html message to be shown when the user enters an invalid character. For instance, "00:00:00;;;warning=invalid character" displays a "invalid character" tooltip once the user types in invalid character, in this case any character that's not a digit. The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value ( including the literals ). If this option should display/use the semicolon (;) character is should be \; ( escape )
- **invalid=value**, indicates the html message to be displayed when the user enters an inappropriate value for the field. If the value is missing or empty, the option has no effect, so no validation is performed. If the value is a not-empty value, the validation is performed. If the value is single space, no message is displayed

and the field is keep opened while the value is inappropriate. For instance, "!(999) 000 0000;;;invalid=The value you entered isn't appropriate for the input mask <b>'<%mask%>'</b> specified for this field." displays the "The value you entered isn't appropriate for the input mask '...' specified for this field." tooltip once the user leaves the field and it is not-valid ( for instance, the field includes entities required and uncompleted ). The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value ( including the literals ). If this option should display/use the semicolon (;) character is should be \; ( escape ). This option can be combined with empty, validateas.

- **validateas=value**, specifies the additional validation is done for the current field. If value is missing or 0 (exValidateAsNone), the option has no effect. The validateas option has effect only if the invalid option specifies a not-empty value. Currently, the value can be 1 (exValidateAsDate), which indicates that the field is validated as a date. For instance, having the mask "!(00/00/0000;;0;empty,validateas=1,invalid=Invalid date!,warning=Invalid character!,select=4,overtyp", indicates that the field is validate as date ( validateas=1 ).
- **empty**, indicates whether the field supports empty values. This option can be used with invalid flag, which indicates that the user can leave the field if it is empty. If empty flag is present, the field displays nothing if no entity is completed ( empty ). Once the user starts typing characters the current mask is displayed. For instance, having the mask "!(999) 000 0000;;;empty,select=4,overtyp,invalid=invalid phone number,beep", it specifies an empty or valid phone to be entered.
- **select=value**, indicates what to select from the field when it got the focus. The value could be 0 ( nothing, exSelectNoGotFocus ), 1 ( select all, exSelectAllGotFocus ), 2 ( select the first empty and editable entity of the field, exSelectEditableGotFocus ), 3 ( moves the cursor to the beginning of the first empty and editable entity of the field, exMoveEditableGotFocus ), 4 ( select the first empty, required and editable entity of the field, exSelectRequiredEditableGotFocus ), 5 ( moves the cursor to the beginning of the first empty, required and editable entity of the field, exMoveRequiredEditableGotFocus ). For modes 2 and 4 the entire field is selected if no matching entity is found. For instance, "Time:`XX:XX;;;select=1" indicates that the entire field ( including the Time: prefix ) is selected once it get the focus. The "Time:`XX:XX;;;select=3", moves the cursor to first X, if empty, the second if empty, and so on

**Experimental:**

**multiline**, specifies that the field supports multiple lines.

**rich**, specifies that the field displays a rich type editor. By default, the standard edit field is shown

**disabled**, shows as disabled the field.

## ContextMenu - Masking Float

The [mask=<VALUE>] property may indicate the followings, if the [float=-1] is present

- **negative number**: if the first character in the mask is - ( minus ) the control supports negative numbers. Pressing the - key will toggle the sign of the number. The + sign is never displayed.
- **decimal symbol**: the last character that's different than # ( digit ), or 0 (zero) indicates the decimal symbol. If it is not present the control mask a floating point number without decimals.
- **thousand symbol**: the thousand symbol is the last character that's not a # ( digit ), 0 (zero) or it is not the decimal symbol as explained earlier, if present.
- the maximum **number of decimals** in the number ( the # or 0 character after the decimal symbol )
- the maximum number of digits in the integer part ( the number of # or 0 character before decimal symbol )
- the **0** character indicates a **leading-zero**. The count of 0 (zero) characters before decimal character indicates the leading-zero for integer part of the control, while the count of 0 (zero) characters after the decimal separator indicates the leading-zero for decimal part of the control. For instance, the Mask on "-###,###,##0.00", while the control's Text property is 1, the control displays 1.00, if 1.1 if displays 1.10, and if empty, the 0.00 is displayed.

If the <VALUE> property is empty, the control takes the settings for the regional options like: Decimal Symbol , No. of digits after decimal, Digit grouping symbol.

Here are few samples:

The <VALUE>"-###.###.##0,00" filter floating point numbers a number for German settings ( "," is the decimal sign, "." is the thousands separator ). This format displays leading-zeros.

The <VALUE>"-###.###.###,##" filter floating point numbers a number for German settings ( "," is the decimal sign, "." is the thousands separator )

The <VALUE>"-###,###,###.##" filter floating point numbers a number for English settings ( "." is the decimal sign, "," is the thousands separator )

The <VALUE>"####" indicates a max-4 digit number ( positive ) without a decimal symbol and without digit grouping

The <VALUE>"-###.#" filters a floating point number from the -99.9 to 99.9 ( "." is the decimal sign, no thousands separator )

The <VALUE>"#,###.##" filters a floating point number from the 0 to 9,999.99 with digit grouping ( "." is the decimal sign, "," is the thousands separator ).

ContextMenu - Tick Label Expression



For instance:

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

The The <VALUE> of [ticklabel] option is a formatted expression which result may include the [HTML](#) tags.

The The <VALUE> of [ticklabel] option indicates a formatting expression that may use the following predefined keywords:

- **value** gets the slider's position to be displayed
- **current** gets the current slider's value.
- **vmin** gets the slider's minimum value.
- **vmax** gets the slider's maximum value.
- **smin** gets the slider's selection minimum value.
- **smax** gets the slider's selection maximum value.

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

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

***"expression ? true\_part : false\_part"***

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

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

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

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

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

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

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

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

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

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

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

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

*The supported conversion unary operators are:*

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

Here's few predefined types:

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

- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS.

*Other known operators for numbers are:*

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

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

- *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
- *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
- *Grouping* - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the

field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:

- 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
- 1 - Negative sign, number; for example, -1.1
- 2 - Negative sign, space, number; for example, - 1.1
- 3 - Number, negative sign; for example, 1.1-
- 4 - Number, space, negative sign; for example, 1.1 -
- **LeadingZero** - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

*Other known operators for strings are:*

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

*Other known operators for dates are:*

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.

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

The The <VALUE> of [ticklabel] option can display labels using the following built-in HTML tags:

- **<b></b>** displays the text in **bold**.
- **<i></i>** displays the text in *italics*.
- **<u></u>** underlines the text.
- **<s></s>** Strike-through text
- **<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>** displays text with a specified **foreground** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bgcolor=RRGGBB></bgcolor>** displays text with a specified **background** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<br>** a forced line-break
- **<solidline>** The next line shows a solid-line on top/bottom side. If has no effect for a single line caption.
- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single line caption.
- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text

- **<c>** Centers the text
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&amp;**; ( & ), **&lt;**; ( < ), **&gt;**; ( > ), **&qout;** ( " ) and **&#number** ( the character with specified code ), For instance, the **&#8364** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **<b>bold</b>** in HTML caption you can use **&lt;b&gt;bold&lt;/b&gt;**;

# property Surface.ElementFormat as String

Specifies the way the control shows the parts of the elements.

Type	Description
String	A String format that specified the <a href="#">CRD</a> format to arrange the parts of the element. This setting is applied to all elements that has the <a href="#">ElementFormat</a> property on empty

By default, the ElementFormat property is empty. Use the ElementFormat property to arrange parts of the elements in a different way. The [ElementFormat](#) property specifies the format to display parts of a specified element. The ElementFormat property is applied to all elements with the [ElementFormat](#) property on empty.

The know parts of the element are:

- **check**, specifies the part where the element's checkbox is displayed. The [ShowCheckBox](#) property specifies whether the element's checkbox is shown or hidden. *The [CheckBoxAlign](#) property specifies the alignment of the checkbox relative to the "check" part of the element.*
- **caption**, specifies the part where the element's caption is displayed. The [Caption](#) property specifies the element's HTML caption. *The [CaptionAlign](#) property specifies the alignment of the caption relative to the "caption" part of the element.*
- **extracaption**, specifies the part where the element's extra-caption is displayed. The [ExtraCaption](#) property specifies the element's HTML extra-caption. *The [ExtraCaptionAlign](#) property specifies the alignment of the extra-caption relative to the "extracaption" part of the element.*
- **picture**, specifies the part where the element's pictures are displayed. The [Pictures](#) property specifies the element's pictures. *The [PicturesAlign](#) property specifies the alignment of the pictures relative to the "picture" part of the element.*
- **extrapicture**, specifies the part where the element's extra-pictures are displayed. The [ExtraPictures](#) property specifies the element's extra-pictures. *The [ExtraPicturesAlign](#) property specifies the alignment of the extra-pictures relative to the "extrapicture" part of the element.*
- **client**, specifies the part of the element where the inside ActiveX is displayed. The [Control](#) property indicates the inside ActiveX that hosted by the element.

The parts of the elements must be included between "" in order to be recognized by the [CRD](#) format.

For instance:

- "check" indicates that just the element's checkbox is displayed, so no matter if other



are set the element displays just the checkbox ( if [ShowCheckBox](#) property is True ) and the element's Caption property.

- "check,caption" indicates that just the element's checkbox and caption are displayed, so no matter if other are set the element displays just the checkbox ( if [ShowCheckBox](#) property is True ) and the element's [Caption](#) property.
- "client" specifies that the whole element displays just the client part, so the inside Active control will use the entire background to display the inside ActiveX control. This indicates, that no caption, check or any other part is displayed on the element.
- "check":18,"client", displays the element's checkbox aligned to the left on a 18-pixels wide, and displays the client on the rest of the element.
- "18;"caption"/"client"", allows the element's [Caption](#) and the [Control](#) to be displayed.

property Surface.ElementFromPoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS) as Element

Gets the Element object from the cursor.

Type	Description
X as OLE_XPOS_PIXELS	A Long expression that specifies the x-cursor position.
Y as OLE_YPOS_PIXELS	A Long expression that specifies the y-cursor position.
<a href="#">Element</a>	An Element object from the cursor

The ElementFromPoint(-1,-1) property returns the element from the cursor or nothing if no element at the **current cursor position**. The [ElementFromPosition](#) property determines the element from the surface giving the coordinates on the surface. Use the [LinkFromPoint](#) property to get if there is any link at the specified position. The [HitTestFromPoint](#)(-1,-1) property returns the element and the hit-test code from the cursor. The [ShowHandCursorOn](#) property specifies the parts of the element that shows the hand cursor when the mouse-pointer hovers the part. The [HitTestFromPoint](#)(-1,-1) property returns the element/hit-test code from the cursor.

# property Surface.ElementFromPosition (X as Long, Y as Long) as Element

Gets the Element object from the position.

Type	Description
X as Long	A Long expression that specifies the x-coordinate on the surface.
Y as Long	A Long expression that specifies the y-coordinate on the surface.
<a href="#">Element</a>	An Element object from the position.

The (0,0) point indicates the origin of the surface, so the X and Y parameters must be relative to the origin of the surface. The ElementFromPoint property gets the element from the surface giving the position on the surface. Use the [PointToPosition](#) / [PositionToPoint](#) property to convert screen coordinates to surface coordinates or reverse. The [ElementFromPoint\(-1,-1\)](#) property returns the element from the cursor or nothing if no element at the **current cursor position**. The [X](#) and [Y](#) properties specifies the position of the element on the surface. The [Width](#) property specifies the width of the element, in surface coordinates. The [Height](#) property specifies the height of the element, in surface coordinates.

# property Surface.Elements as Elements

Retrieves the control's elements.

Type	Description
<a href="#">Elements</a>	The surface's Elements collection.

The Elements property gives the access to the surface's Elements collection. The [Add](#) method adds programmatically a new element to the surface. Use the [Insert](#) method to insert programmatically a child element. Use the [InsertControl](#) method to insert programmatically a child element that hosts an inner ActiveX control. The control fires the [AddElement](#) event once a new element is added to the Elements collection. The [AllowCreateObject](#) property specifies the combination of keys that allows the user to create objects on the surface. The [Links](#) property gives access to the surface's Links collection.

# property Surface.Enabled as Boolean

Enables or disables the control.

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

By default, the Enabled property is True. Use the Enabled property of the control to disable the entire surface. The Enabled property changes the Enabled state of the window that hosts the surface. While the control's Enabled property is False, no mouse or key events are generated. Use the [Enabled](#) property to show the element as disabled ( grayed ). The [Selectable](#) property specifies whether the user can select the element at runtime. The [Resizable](#) property specifies whether the element can be resized at runtime.

# method Surface.EndBlockUndoRedo ()

Ends recording the UI operations and adds the undo/redo operations as a block, so they all can be restored at once, if Undo method is performed.

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

You can use the [StartBlockUndoRedo](#) / EndBlockUndoRedo methods to group multiple Undo/Redo operations into a single-block. The GroupUndoRedoActions groups the next to current Undo/Redo Actions in a single block. A block may hold multiple Undo/Redo actions. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. Use the GroupUndoRedoActions method to group two or more entries in the Undo/Redo queue in a single block, so when a next Undo/Redo operation is performed, multiple actions may occur. For instance, moving several elements in the same time ( multiple elements selection ) is already recorded as a single block. Use the [UndoRedoQueueLength](#) property to specify the number of entries that Undo/Redo queue may store.

A block starts with StartBlock and ends with EndBlock when listed by [UndoListAction](#)/[RedoListAction](#) property as in the following sample:

```
StartBlock
MoveElement;B
MoveElement;A
EndBlock
```

# method Surface.EndUpdate ()

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

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

Use the [Refresh](#) method to refresh the control.

# property Surface.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 Surface.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, you can use the EXPRINT.PrintExt = CONTROL.ExecuteTemplate("me") to print the control's content.

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

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

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

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

An instruction can be one of the following:

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

The Template supports the following general functions:

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

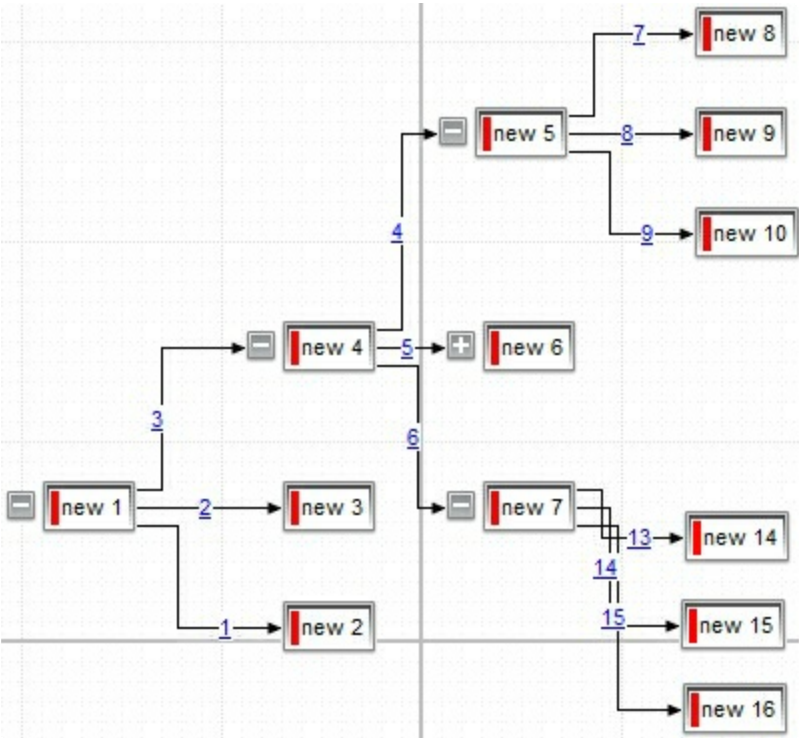
# property Surface.ExpandLinkedElements as Boolean

Specifies whether the linked elements are expanded or collapsed.

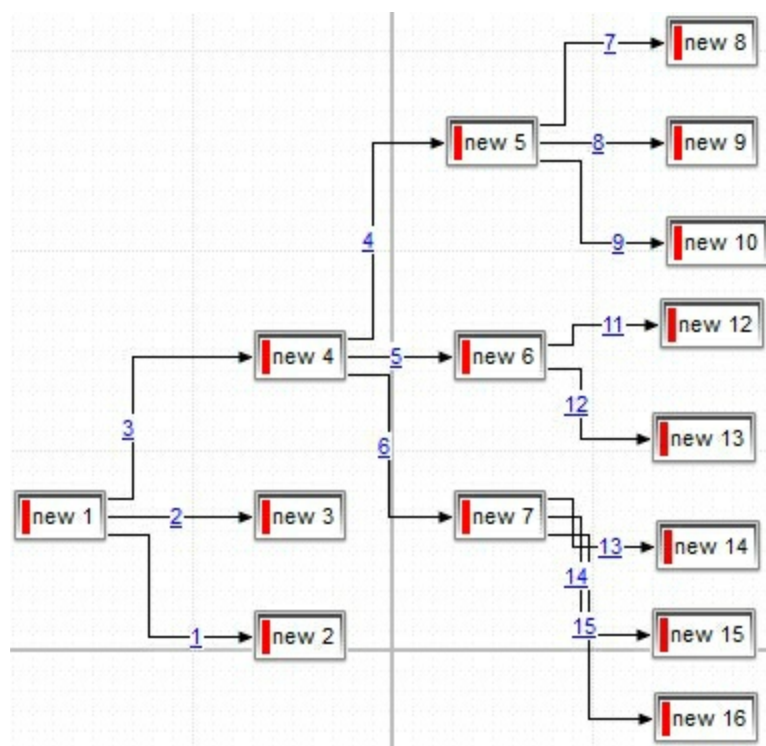
Type	Description
Boolean	A Boolean expression that specifies whether the control expands or collapses the descendent elements. By descending element of current element we mean any element that has a link that starts from current element and ends to this element.

By default, the ExpandLinkedElements property is False. Use the ExpandLinkedElements property to add expand/collapse glyphs next to elements that has outgoing links. The [OutgoingLinks](#) property specifies the links that starts from the element. The [Expanded](#) property specifies whether the element is expanded or collapsed. Use the [AllowMoveDescendents](#) property to specify whether all descendents elements are moved once the focused element is moved. The [Add](#) method adds programmatically a link between two elements. Use the [ShowLinksOnCollapse](#) property to show the links between an element and collapsed elements. The [ShowLinks](#) property specifies the way the control shows the links on the surface.

The following screen shot shows the surface with ExpandLinkedElements property is True:



The following screen shot shows the surface with ExpandLinkedElements property is False ( by default ):



# method Surface.FitToClient ()

Resizes or/and moves the entire chart to fit the control's client area.

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

# property Surface.FocusLink as Variant

Gets or sets the focused link

Type	Description
	<p>The FocusLink property is a get/set property that returns:</p> <ul style="list-style-type: none"><li>• empty value, no link has been clicked/focused</li><li>• a <a href="#">Link</a> object that defines the link being focused</li></ul> <p>or it can be:</p>
Variant	<ul style="list-style-type: none"><li>• an empty value, to clear the focused link (no link is focused)</li><li>• a string expression that determines the key of the newly focused link</li><li>• a numeric expression that specifies the index of the new link with the focus</li><li>• a <a href="#">Link</a> object that defines the newly focused link</li></ul>

By default, the FocusLink property is empty. The FocusLink property retrieves or changes the current link that is currently focused (selected or active) within the control. Getting the focused link might be useful if you want to perform an action based on the user's current selection. Setting the focused link might be useful if you want to guide the users navigation programmatically (e.g., moving focus to a specific link after a user action). The [LayoutStartChanging\(exFocusLink\)](#) / [LayoutEndChanging\(exFocusLink\)](#) property notifies your application once the user focuses a new link.

# property Surface.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object used to paint the elements.

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 columns or items.

The following VB sample assigns by code a new font to the control:

```
With Surface1
    With .Font
        .Name = "Tahoma"
    End With
    .Refresh
End With
```

The following C++ sample assigns by code a new font to the control:

```
COleFont font = m_surface.GetFont();
font.SetName( "Tahoma" );
m_surface.Refresh();
```

the C++ sample requires definition of COleFont class ( #include "Font.h" )

The following VB.NET sample assigns by code a new font to the control:

```
With AxSurface1
    Dim font As System.Drawing.Font = New System.Drawing.Font("Tahoma", 10,
    FontStyle.Regular, GraphicsUnit.Point)
    .Font = font
    .CtlRefresh()
End With
```

The following C# sample assigns by code a new font to the control:

```
System.Drawing.Font font = new System.Drawing.Font("Tahoma", 10, FontStyle.Regular);
axSurface1.Font = font;
```



```
axSurface1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the control:

```
with thisform.Surface1.Object  
    .Font.Name = "Tahoma"  
    .Refresh()  
endwith
```

The following Template sample assigns by code a new font to the control:

```
Font  
{  
    Name = "Tahoma"  
}
```

# property Surface.ForeColor as Color

Specifies the control's foreground color.

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

The ForeColor property defines the control's foreground color.. The [Picture](#) property to assign your logo on the control's background. The control uses the [PictureDisplay](#) property to determine how the picture is displayed on the control's background. The [Background](#)(exElementForeColor) property specifies the default element's foreground color. The [BackColor](#) property specifies the control's background color.

# method Surface.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 Surface.

The FormatABC method formats the A,B,C values based on the giving expression and returns the result.

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 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

This property/method supports predefined constants and operators/functions as described [here](#).

# property Surface.FormatAnchor(New as Boolean) as String

Specifies the visual effect for anchor elements in HTML captions.

Type	Description
New as Boolean	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. *You can use the <a> anchor elements to insert hyperlinks to cells, bars or links.* Use the [Caption](#) property to specify the element's caption.

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 Surface.FreezeEvents (Freeze as Boolean)

Prevents the control to fire any event.

Type	Description
Freeze as Boolean	A Boolean expression that specifies whether the control's 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 improve performance of the control while loading data into it.

## Purpose:

- FreezeEvents(True) is used to temporarily stop (or "freeze") a control from responding to any events (such as clicks, changes in value, etc.).
- FreezeEvents(False) re-enables (or "unfreezes") event handling for the control, allowing it to respond to events again.

## Use Case:

- Imagine you're making multiple updates to a control, such as a list, a form, or a UI component, and you don't want the control to react to each individual change (e.g., trigger event handlers after each modification). You would use FreezeEvents(True) to pause event handling while making the updates, then use FreezeEvents(False) once all updates are complete.

## Example Scenario:

- Lets say you have a list where changing the selection triggers an event. You want to programmatically add several items to the list, but you don't want the selection-changed event to fire every time you add a new item:

```
control.FreezeEvents(True) // Stop event processing
// Perform multiple changes to the control
control.AddItem("Item 1")
control.AddItem("Item 2")
control.AddItem("Item 3")
control.FreezeEvents(False) // Resume event processing
```

Without freezing the events, the control might trigger its event handler each time an item is added. Freezing prevents that, ensuring that the control remains "quiet" during updates.

## Benefits:

- Improved performance: Prevents unnecessary event handling during batch updates.  
Avoids unintended side effects: Stops event handlers from running when you don't want them to (e.g., while setting up or modifying the control).

# method Surface.GroupUndoRedoActions (Count as Long)

Groups the next to current Undo/Redo Actions in a single block.

Type	Description
Count as Long	A Long expression that specifies the number of entries being grouped in a single block of actions, in the Undo/Redo queue.

The GroupUndoRedoActions groups the next to current Undo/Redo Actions in a single block. A block may hold multiple Undo/Redo actions. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. Use the GroupUndoRedoActions method to group two or more entries in the Undo/Redo queue in a single block, so when a next Undo/Redo operation is performed, multiple actions may occur. You can use the [StartBlockUndoRedo](#) / [EndBlockUndoRedo](#) methods to group multiple Undo/Redo operations into a single-block. For instance, moving several elements in the same time ( multiple elements selection ) is already recorded as a single block. Use the [UndoRedoQueueLength](#) property to specify the number of entries that Undo/Redo queue may store.

A block starts with StartBlock and ends with EndBlock when listed by [UndoListAction](#)/[RedoListAction](#) property as in the following sample:

```
StartBlock
MoveElement;B
MoveElement;A
EndBlock
```

# property Surface.HideSel as Boolean

Returns a value that determines whether selected item appears highlighted when a control loses the focus.

Type	Description
Boolean	A Boolean expression that determines whether selected item appears highlighted when a control loses the focus.

By default, the HideSel property is True, which indicates that selected elements are not highlighted when the control loses the focus. The HideSel property specifies whether the selected elements are highlighted or not when the control loses the focus. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [Selectable](#) property of the Element object indicates whether the element is selectable or unselectable. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. Use the [AllowMoveSelection](#) property to prevent moving the entire selection when focused element is moved. Use the [AllowResizeSelection](#) property to prevent resizing the entire selection when focused element is resized. Use the [AllowMoveDescendents](#) property to prevent moving all descendents ( children and outgoing elements ) when focused element is moved.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.



# property Surface.HitTestFromPoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS) as HitTest

Gets the Element object and the Hit-Test code from the cursor.

Type	Description
X as OLE_XPOS_PIXELS	A Long expression that specifies the x-cursor position.
Y as OLE_YPOS_PIXELS	A Long expression that specifies the y-cursor position.
<a href="#">HitTest</a>	A HitTest object that holds information about the Element from the specified position.

The HitTestFromPoint property returns the element/hit-test code from the specified position. The HitTestFromPoint(-1,-1) property returns the element/hit-test code from the current cursor position. The [ElementFromPoint](#) property returns the element from the cursor. For instance, you can use the HitTestFromPoint property to determine whether the cursor hovers the expand/collapse glyphs, the element's checkbox, picture and so on.

The following VB sample determines if the cursor hovers the element's expand/collapse glyphs:

```
Private Sub Surface1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim h As HitTest
    Set h = Surface1.HitTestFromPoint(-1, -1)
    If Not h Is Nothing Then
        If (h.HitTestCode And exHitTestMask) = exHitTestGlyph Then
            Debug.Print "Expand/Collapse Glyph of " & h.Element.ID
        End If
    End If
End Sub
```

# method Surface.Home ()

Restores the view to the origin.

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

The Home method moves the surface's scroll position to 0 and the zooming factor to 100%. The control fires the [LayoutStartChanging](#)(exSurfaceHome) / [LayoutEndChanging](#)(exSurfaceHome) event when the user clicks the Home button. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position. The [Zoom](#) property specifies the current zooming factor.

# property Surface.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none"><li>• a string expression that indicates the path to the picture file, being loaded.</li><li>• a string expression that indicates the base64 encoded string that holds a picture object, Use the <a href="#">eximages</a> tool to save your picture as base64 encoded format.</li><li>• A Picture object that indicates the picture being added or replaced. ( A Picture object implements IPicture interface ),</li></ul> <p>If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added.</p>

By default, the HTMLPicture collection is empty. The HTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the <img> tags. 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 [Images](#) method specifies the list of 16x16 icons to be displayed on the control's surface. The [Caption](#) property specifies the caption of the element ( including icons, picture and so on ). Use the [Pictures](#) / [ExtraPictures](#) properties to display different pictures on the element.

# property Surface.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 Surface.Images (Handle as Variant)

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

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

The Handle parameter can be:

- A string expression that specifies the ICO file to add. The ICO file format is an image file format for computer icons in Microsoft Windows. ICO files contain one or more small images at multiple sizes and color depths, such that they may be scaled appropriately. For instance, Images("c:\temp\copy.ico") method adds the sync.ico file to the control's Images collection (*string, loads the icon using its path*)
- A string expression that indicates the BASE64 encoded string that holds the icons list. Use the Exontrol's [ExImages](#) tool to save/load your icons as BASE64 encoded format. In this case the string may begin with "gBJJ..." (*string, loads icons using base64 encoded string*)
- A reference to a Microsoft ImageList control (mscomctl.ocx, MSComctlLib.ImageList type) that holds the icons to add (*object, loads icons from a Microsoft ImageList control*)
- A reference to a Picture (IPictureDisp implementation) that holds the icon to add. For instance, the VB's LoadPicture (Function LoadPicture([FileName], [Size], [ColorDepth], [X], [Y]) As IPictureDisp) or LoadResPicture (Function LoadResPicture(id, restype As Integer) As IPictureDisp) returns a picture object (*object, loads icon from a Picture object*)
- A long expression that identifies a handle to an Image List Control ( the Handle should be of HIMAGELIST type ). On 64-bit platforms, the Handle parameter must be a Variant of LongLong / LONG\_PTR data type ( signed 64-bit (8-byte) integers ), saved under lVal field, as VT\_I8 type. The LONGLONG / LONG\_PTR is \_\_int64, a 64-bit integer. For instance, in C++ you can use as Images( COleVariant( (LONG\_PTR)hImageList) ) or Images( COleVariant(

Handle as Variant

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

---

The Images method assigns a list of icons to be displayed on the control's surface. The icons can be displayed on the control's using the <img>number</img> HTML tags. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. The [HTMLPicture](#) property handles a collection of custom size picture being displayed in the HTML captions, using the <img>key</img> tags. The [Replacelcon](#) method replaces icons in the control's . The [Caption](#) property specifies the caption of the element ( including icons, picture and so on ). Use the [Pictures](#) / [ExtraPictures](#) properties to display different pictures on the element.

# property Surface.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.

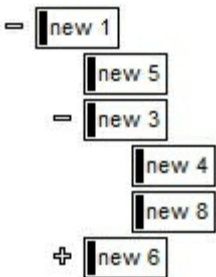
# property Surface.IndentX as Long

Specifies the child elements indentation on x-axis.

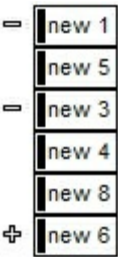
Type	Description
Long	A Long expression that specifies the child elements indentation on x-axis.

By default, the IndentX is 24 pixels. Use the IndentX property to specify the indentation between child and parent elements on x-axis. The [IndentY](#) property specifies the indentation between children and parent elements on y-axis. The [Insert](#) method add programmatically a child element. The [Expanded](#) property expands or collapse the parent element.

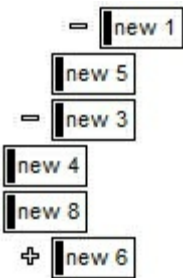
The following screen shot shows the tree using the default values for IndentX(24), IndentY(2) properties:



The following screen shot shows the tree using the IndentX(0), IndentY(0) properties:



The following screen shot shows the tree using the IndentX(-24), IndentY(2) properties:





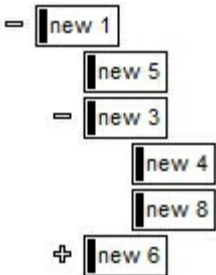
# property Surface.IndentY as Long

Specifies the child elements indentation on y-axis.

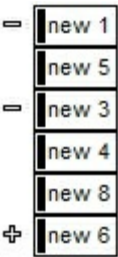
Type	Description
Long	A Long expression that specifies the child elements indentation on y-axis.

By default, the IndentY is 2 pixels. The IndentY property specifies the indentation between children and parent elements on y-axis. Use the [IndentX](#) property to specify the indentation between child and parent elements on x-axis. The [Insert](#) method add programmatically a child element. The [Expanded](#) property expands or collapse the parent element.

The following screen shot shows the tree using the default values for IndentX(24), IndentY(2) properties:



The following screen shot shows the tree using the IndentX(0), IndentY(0) properties:



The following screen shot shows the tree using the IndentX(-24), IndentY(2) properties:



## property Surface.Layout as String

Saves or loads the control's layout, such as positions of the columns, scroll position, filtering values.

Type	Description
String	A String expression that specifies the control's layout.

You can use the Layout property to store the control's layout and to restore the layout later. For instance, you can save the control's Layout property to a file when the application is closing, and you can restore the control's layout when the application is loaded. The Layout property saves almost all of the control's properties that user can change at runtime ( like changing the column's position by drag and drop ). The Layout property does NOT save the control's data, so the Layout property should be called once you loaded the data from your database, xml or any other alternative. Once the data is loaded, you can call the Layout property to restore the View as it was saved. Before closing the application, you can call the Layout property and save the content to a file for reading next time the application is opened.

The Layout property saves/loads the following information:

- surface's scroll position as indicated by [ScrollX](#) and [ScrollY](#) properties.
- surface's zoom factor as indicated by the [Zoom](#) property.
- selected elements as indicated by the [Selection](#) property

These properties are serialized to a string and encoded in BASE64 format.

The following movies show how Layout works:

-  The Layout property is used to save and restore the control's view.

# property Surface.LinkFromPoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS) as Link

Gets the Link object from the cursor.

Type	Description
X as OLE_XPOS_PIXELS	A Long expression that specifies the x-cursor position.
Y as OLE_YPOS_PIXELS	A Long expression that specifies the y-cursor position.
<a href="#">Link</a>	A Link object from the cursor

Use the LinkFromPoint property to get if there is any link at the specified position. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#)(-1,-1) property returns the element and the hit-test code from the cursor. The [ShowHandCursorOn](#) property specifies the parts of the element that shows the hand cursor when the mouse-pointer hovers the part. The [HitTestFromPoint](#)(-1,-1) property returns the element/hit-test code from the cursor.

# property Surface.Links as Links

Retrieves the control's links.

Type	Description
<a href="#">Links</a>	A Links object that indicates the surface's Links collection.

The Links property gives the access to the surface's Links collection. The [Add](#) method adds programmatically a Link object to the collection and returns a reference to the newly created object. The [AllowLinkObjects](#) property specifies the combination of keys that allows the user to link the objects. The [AddLink](#) event notifies your application once a new link is added to the Links collection.

# property Surface.LinksArrowColor as Color

Specifies the color/visual appearance to draw the arrows of the links between the elements.

Type	Description
Color	A Color expression that defines the color to show the arrow of the links. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used to paint the part. Use the <a href="#">Add</a> method to add new skins to the control. In other words, you can use the EBN objects to define a different type of arrows.

By default, the LinksArrowColor is -1, which indicates that the [LinksColor](#) property specifies the color of the arrow. The LinksArrowColor property specifies the color to show the arrow of the links. The [ArrowColor](#) property specifies the color to show the direction of the particular link. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links.

# property Surface.LinksArrowFrameColor as Color

Specifies the color to show the default frame of the arrow

Type	Description
Color	A Color expression to show the arrow's frame

By default, the LinksArrowFrameColor property is -1 which indicates that [LinksArrowColor](#) property specifies the color of the arrow's frame. The LinksArrowFrameColor property specifies the color to show the default frame of the arrow. The [ArrowFrameColor](#) property specifies the color to show the arrow's frame for a particular link. Use the [LinksColor](#) property to define the color to show all links on the surface. The [Color](#) property specifies the color for an individual link. The [LinksArrowColor](#) property specifies the color to show the arrow of the links. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [LinksArrowSize](#) property specifies the size to show the arrow for links.

# property Surface.LinksArrowSize as Long

Specifies the size to show the arrow for links

Type	Description
Long	A long expression that specifies the size of the arrow

By default, the LinksArrowSize property is -1 which indicates that [LinksWidth](#) property controls the size of the arrow. The LinksArrowSize property specifies the size to show the arrow for links. The [ArrowSize](#) property specifies the size to show the arrow for a particular link. Use the [LinksColor](#) property to define the color to show all links on the surface. The [Color](#) property specifies the color for an individual link. The [LinksArrowColor](#) property specifies the color to show the arrow of the links. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [LinksArrowFrameColor](#) property specifies the color to show the default frame of the arrow.

# property Surface.LinksColor as Color

Specifies the color to draw the links between the elements.

Type	Description
Color	A Color expression that specifies the color to show the links between elements on the surface.

By default, the LinksColor property is RGB(0,0,0). Use the LinksColor property to define the color to show all links on the surface. The [Color](#) property specifies the color for an individual link. The [LinksArrowColor](#) property specifies the color to show the arrow of the links. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The [LinksArrowFrameColor](#) property specifies the color to show the default frame of the arrow. The [LinksArrowSize](#) property specifies the size to show the arrow for links.



# property Surface.LinksShowDir as Boolean

Specifies whether the links show or hide the direction/arrow.

Type	Description
Boolean	A Boolean expression that specifies that the arrow for all links are shown or hidden.

The LinksShowDir property specifies whether the direction for all links are shown or hidden. The [ShowDir](#) property specifies whether the arrow of the link is shown or hidden. The [ArrowColor](#) property specifies the color to show the direction of the particular link. The [LinkWidth](#) property specifies the size of the link and so the size of the arrow. The [Visible](#) property indicates whether the link is visible or hidden.

# property Surface.LinksStyle as LinkStyleEnum

Specifies the style to draw the links between the elements.

Type	Description
<a href="#">LinkStyleEnum</a>	A LinkStyleEnum expression that defines the style of the line to be shown between elements.

The LinksStyle property specifies the style of the lines/links to be shown on the surface. The [Style](#) property indicates the style of the line to be shown on a particular link. The [ShowLinksType](#) property defines the type of the link to be shown between elements on the surface. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The control's [LinksStyle](#) property defines the style of the line to be shown on the link. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden.

# property Surface.LinksWidth as Long

Specifies the width in pixels of the pen to draw the links between the elements.

Type	Description
Long	A Long expression that specifies the width to show the links between elements.

The LinksWidth property specifies the width of the links between elements. The [Width](#) property specifies the size of the link and so the size of the arrow. The [ShowLinksType](#) property defines the type of the link to be shown between elements on the surface. The [LinksStyle](#) property specifies the style of the lines/links to be shown on the surface. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden.

# method Surface.LoadXML (Source as Variant)

Loads an XML document from the specified location, using MSXML parser.

Type	Description
Source as Variant	An indicator of the object that specifies the source for the XML document. The object can represent a file name, a URL, an IStream, a SAFEARRAY, or an IXMLDOMDocument.
Return	Description
Boolean	A boolean expression that specifies whether the XML document is loaded without errors. If an error occurs, the method retrieves a description of the error occurred.

The LoadXML method uses the MSXML ( MSXML.DOMDocument, XML DOM Document ) parser to load XML documents, previously saved using the [SaveXML](#) method. The LoadXML method loads elements and links. The elements of the surface go to the **<Elements>** section, while the links go to the **<Links>** section. Each element goes under the **<element>** section, while each information about a link goes to the **<link>** section.

The [XML Format](#) of the file is:

```
- <Content Author Component Version ...>
  - <Elements>
    <Element ID ... />
    ...
  </Elements>
  - <Links>
    <Link ID ...>
    </Link>
  </Links>
</Content>
```

# property Surface.MajorGridColor as Color

Indicates the color to show the major grid lines on the surface.

Type	Description
Color	A Color expression that specifies the color to show the major grid lines.

The MajorGridColor property specifies the color to show the major grid lines. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.MajorGridHeight as Long

Indicates the height between two consecutive major grid lines.

Type	Description
Long	A Long expression that specifies the distance (in pixels ) between two consecutive major grid lines.

Use the [MajorGridWidth](#) / MajorGridHeight property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.MajorGridStyle as LinesStyleEnum

Specifies the style to display the major grid lines.

Type	Description
<a href="#">LinesStyleEnum</a>	A LinesStyleEnum expression that specifies the style of major grid lines.

By default, the MajorGridStyle property is exLinesDot. Use the MajorGridStyle property to specify the style of the major lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.MajorGridWidth as Long

Indicates the width between two consecutive major grid lines.

Type	Description
Long	A Long expression that specifies the distance (in pixels ) between two consecutive major grid lines.

Use the MajorGridWidth / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.



# property Surface.MinorGridColor as Color

Indicates the color to show the minor grid lines on the surface.

Type	Description
Color	A Color expression that specifies the color to show the minor grid lines.

The MinorGridColor property specifies the color to show the minor grid lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.MinorGridHeight as Long

Indicates the height between two consecutive minor grid lines.

Type	Description
Long	A Long expression that specifies the distance (in pixels ) between two consecutive minor grid lines.

Use the [MinorGridWidth](#) / MinorGridHeight property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.MinorGridStyle as LinesStyleEnum

Specifies the style to display the minor grid lines.

Type	Description
<a href="#">LinesStyleEnum</a>	A LinesStyleEnum expression that specifies the style to show the minor lines.

By default, the MinorGridStyle property is exLinesDot4. Use the MinorGridStyle property to specify the style of the minor lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.MinorGridWidth as Long

Indicates the width between two consecutive minor grid lines.

Type	Description
Long	A Long expression that specifies the distance (in pixels ) between two consecutive minor grid lines.

Use the MinorGridWidth / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [ShowGridLines](#) property to specify whether the control shows or hides the minor/major grid lines. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# method Surface.MoveCorner (From as ContentAlignmentEnum, To as ContentAlignmentEnum)

Moves or scrolls the surface.

Type	Description
From as <a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that indicates the point to move the surface from.
To as <a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that indicates the point to move the surface to.

The MoveCorner method scrolls the surface from a corner to another. The [MovePoint](#) method of the control moves the surface from the one point to another. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position. Use the [ScrollTo](#) method of the control to scroll the surface at specified position.

# method Surface.MovePoint (X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS, To as ContentAlignmentEnum)

Moves or scrolls the surface, so the cursor aligns to specified corner.

Type	Description
X as OLE_XPOS_PIXELS	A Long expression that specifies the x-coordinate on the screen to move the surface from.
Y as OLE_YPOS_PIXELS	A Long expression that specifies the y-coordinate on the screen to move the surface from.
To as <a href="#">ContentAlignmentEnum</a>	A ContentAlignmentEnum expression that specifies the corner of the surface to move the surface to.

The MovePoint method of the control moves the surface from the one point to another. The [MoveCorner](#) method scrolls the surface from a corner to another. Use the [ScrollTo](#) method of the control to scroll the surface at specified position. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position.

# method Surface.OLEDrag ()

Causes a component to initiate an OLE drag/drop operation.

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

The method is only for internal use.

# property Surface.OLEDropMode as exOLEDropModeEnum

Returns or sets how a target component handles drop operations

Type	Description
<a href="#">exOLEDropModeEnum</a>	An exOLEDropModeEnum expression that indicates the OLE Drag and Drop mode. 0 means no drag and drop support, 1 means manual support.

*In the /NET Assembly, you have to use the AllowDrop property as explained here:*

- <https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop>

By default, the OLEDropMode property is exOLEDropNone. Curently, the control supports only manual OLE Drag and Drop operation.

See the [OLEStartDrag](#) and [OLEDragDrop](#) events for more details about implementing drag and drop operations into the ExSurface control.



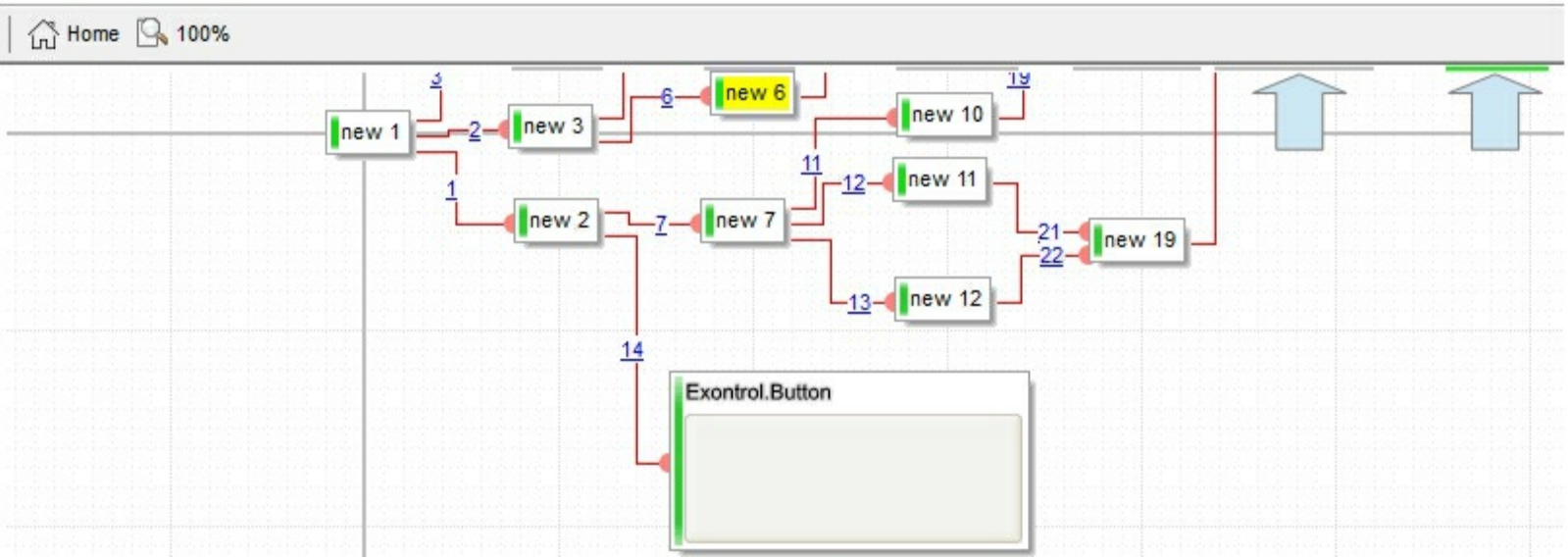
# property Surface.OverviewColor as Color

Specifies the color to show objects outside of the surface's client area.

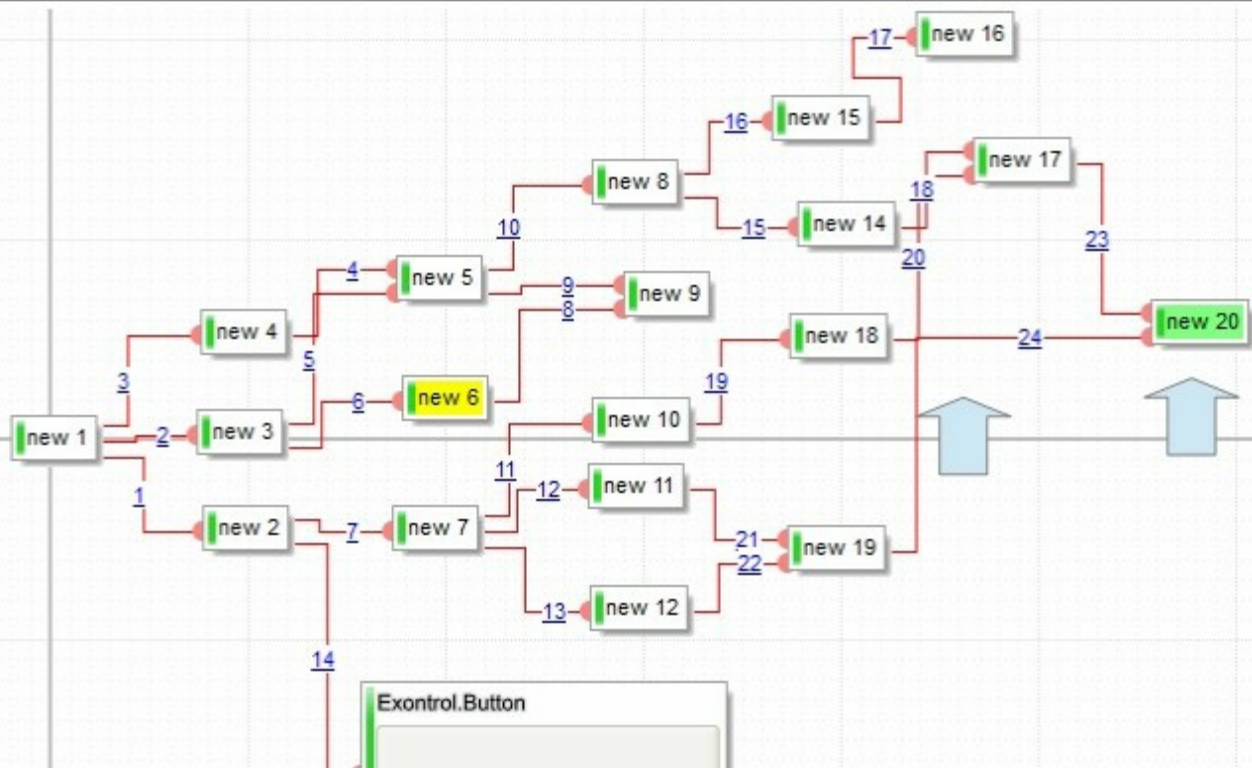
Type	Description
Color	A Color expression that specifies the overview color.

By default, the OverviewColor property is RGB(190,190,190). If the control's OverviewColor property is -1, no elements is shown on the border. Use the [OverviewColor](#) property of the Element to specify a different color to be shown for specific elements. The [OverviewColor](#) property has effect when the element is not fitting the surface's client area and it is shown on the border of the surface. The [BorderWidth](#) / [BorderHeight](#) properties specify the size on the margin where the overview elements are shown.

The following screen shot shows the how elements are shown when they are not visible in the surface's client area ( look on the border ) :



The following screen shot shows the elements when they are visible on the surface's client area:



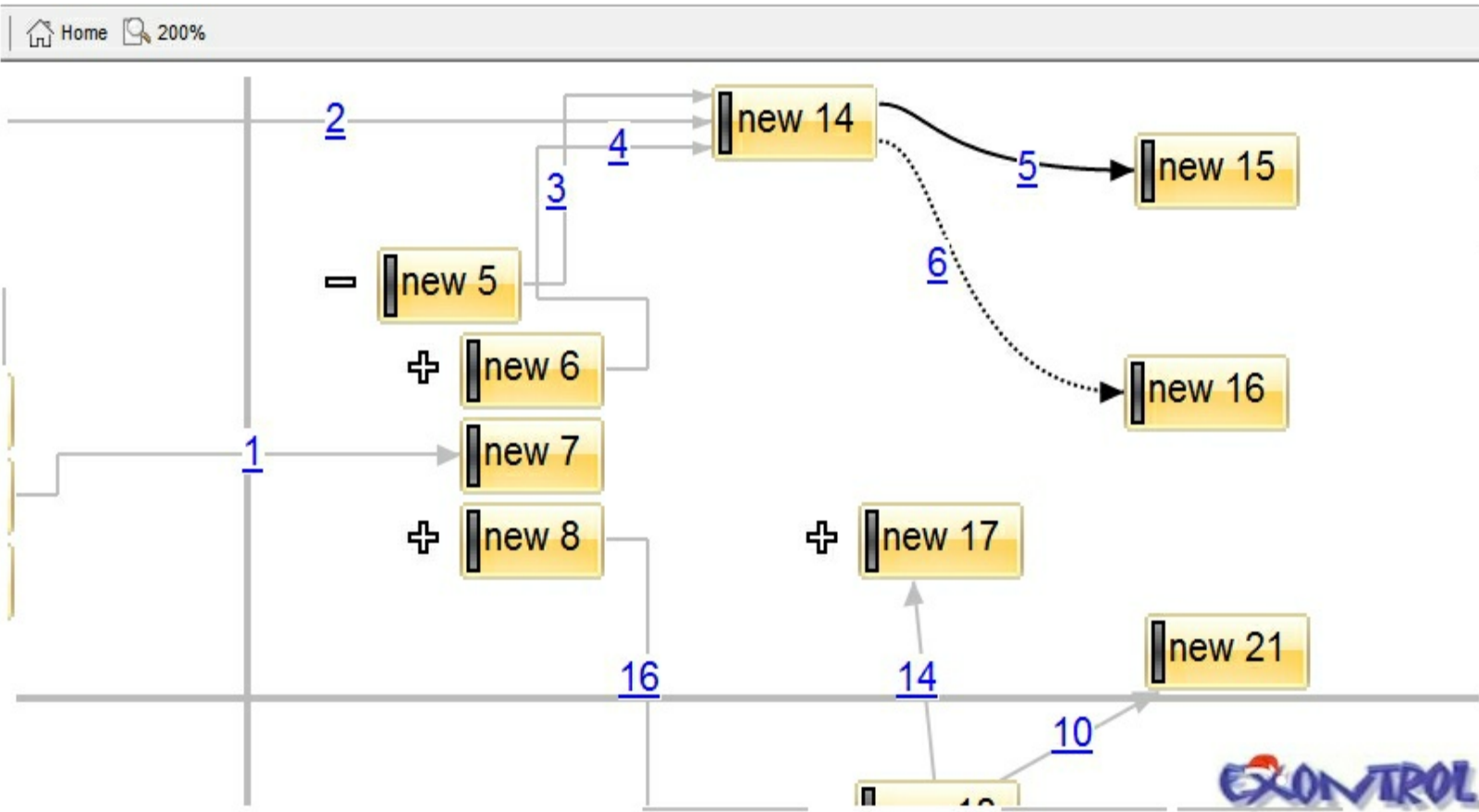
# property Surface.Picture as IPictureDisp

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

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

By default, the control has no picture associated. Use the Picture property to assign your logo on the control's background. The control uses the [PictureDisplay](#) property to determine how the picture is displayed on the control's background. The [BackColor](#) property specifies a solid color to be shown on the control's background. The [Picture](#) property specifies a picture to be displayed on the element's background.

The following screen shot shows a logo on the control's background:



# property Surface.PictureDisplay as PictureDisplayEnum

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

Type	Description
<a href="#">PictureDisplayEnum</a>	A PictureDisplayEnum expression that indicates the way the control arranges the picture on the control's background.

The control uses the PictureDisplay property to determine how the picture is displayed on the control's background. By default, the control has no picture associated. Use the [Picture](#) property to assign your logo on the control's background. The [BackColor](#) property specifies a solid color to be shown on the control's background. The [Picture](#) property specifies a picture to be displayed on the element's background.

## method **Surface.PointToPosition (ByRef X as Long, ByRef Y as Long)**

Converts display coordinates to element's position.

Type	Description
X as Long	(By Reference) A Long expression that determines the x-coordinate on the screen. On return it gets the x-coordinate on the surface.
Y as Long	(By Reference) A Long expression that determines the y-coordinate on the screen. On return it gets the y-coordinate on the surface.

Use the PointToPosition / [PositionToPoint](#) property to convert screen coordinates to surface coordinates or reverse. The PointToPosition / [PositionToPoint](#) convert the giving point based on the [ScrollX](#), [ScrollY](#) and [Zoom](#) conditions ( scroll position and the zooming factor of the surface ). If X and Y are -1, on return the X and Y defines the surface coordinates. The ElementFromPoint property gets the element from the surface giving the position on the surface. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the **current cursor position**. On PowerBuilder, or any other environment that does not support parameters by reference you can use the [ExecuteTemplate](#) method. For instance, you can use a code like ExecuteTemplate("dim x,y;x=-1;y=-1;PointToPosition(x,y);x") that returns the x-position (surface's coordinates) of the current mouse pointer. In the same manner you can use ExecuteTemplate("dim x,y;x=-1;y=-1;PointToPosition(x,y);y") to return the y-position.

## method Surface.PositionToPoint (ByRef X as Long, ByRef Y as Long)

Converts element's position to display coordinates.

Type	Description
X as Long	(By Reference) A Long expression that determines the x-coordinate on the surface. On return it gets the x-coordinate on the screen.
Y as Long	(By Reference) A Long expression that determines the y-coordinate on the surface. On return it gets the y-coordinate on the screen.

Use the [PointToPosition](#) / PositionToPoint property to convert screen coordinates to surface coordinates or reverse. The [PointToPosition](#) / PositionToPoint convert the giving point based on the [ScrollX](#), [ScrollY](#) and [Zoom](#) conditions ( scroll position and the zooming factor of the surface ). The (0,0) point indicates the origin of the surface, so the X and Y parameters must be relative to the origin of the surface. The ElementFromPoint property gets the element from the surface giving the position on the surface. The [ElementFromPoint\(-1,-1\)](#) property returns the element from the cursor or nothing if no element at the **current cursor position**. On PowerBuilder, or any other environment that does not support parameters by reference you can use the [ExecuteTemplate](#) method. For instance, you can use a code like ExecuteTemplate("dim x,y;x=-1;y=-1;PointToPosition(x,y);x") that returns the x-position (surface's coordinates) of the current mouse pointer. In the same manner you can use ExecuteTemplate("dim x,y;x=-1;y=-1;PointToPosition(x,y);y") to return the y-position.

## method Surface.Redo ()

Redoes the next action in the surface's Redo queue.

Type	Description
	The Redo redoes the next action in the control's redo queue. The <a href="#">AllowUndoRedo</a> property enables or disables the Undo/Redo feature. The <a href="#">CanRedo</a> method indicates whether the control can perform a Redo operation. The <a href="#">Undo</a> method undoes the last control operation. The <a href="#">UndoRedoQueueLength</a> property gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue, or in other words how many operations the control's Undo/Redo manager may store.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- **"EndBlock"**, specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked.

# property Surface.RedoListAction ([Action as Variant], [Count as Variant]) as String

Lists the Redo actions that can be performed on the surface.

Type	Description
Action as Variant	<p>[optional] A long expression that specifies the action being listed. If missing or -1, all actions are listed.</p> <p>The Action parameter can be one of the following:</p> <ul style="list-style-type: none"><li>• exUndoRedoAddElement(13) ~ <b>"AddElement;ELEMENTID"</b>, indicates that a new element has been created</li><li>• exUndoRedoRemoveElement(14) ~ <b>"RemoveElement;ELEMENTID"</b>, indicates that an element has been removed</li><li>• exUndoRedoMoveElement(15) ~ <b>"MoveElement;ELEMENTID"</b>, indicates that an element has been moved or resized</li><li>• exUndoRedoUpdateElement(16) ~ <b>"UpdateElement;ELEMENTID"</b>, indicates that one or more properties of the element has been updated, using the <a href="#">StartUpdateElement</a> / <a href="#">EndUpdateElement</a> methods</li><li>• exUndoRedoAddLink(10) ~ <b>"AddLink;LINKID"</b>, indicates that a new link has been created</li><li>• exUndoRedoRemoveLink(11) ~ <b>"RemoveLink;LINKID"</b>, indicates that a link has been removed</li><li>• exUndoRedoUpdateLink(12) ~ <b>"UpdateLink;LINKID"</b>, specifies that one of more properties of the link has been updated, using the <a href="#">StartUpdateLink</a> / <a href="#">EndUpdateLink</a> methods</li></ul> <p>For instance, RedoListAction(12) shows only AddElement actions in the redo stack.</p>
	<p>[optional] A long expression that indicates the number of actions being listed. If missing or -1, all actions are listed. For instance, RedoListAction(12,1) shows only the last AddElement action being added to the redo stack</p>



String

A String expression that lists the Redo actions that may be performed.

The RedoListAction property lists the Redo actions that can be performed in the control. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked. The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- **"EndBlock"**, specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

Here's a sample how the result of RedoListAction method looks like:

```
AddElement;1
UpdateElement;A
AddElement;2
UpdateElement;B
AddLink;1
UpdateLink;Akak
MoveElement;B
StartBlock
MoveElement;3
AddElement;3
```



# method Surface.RedoRemoveAction ([Action as Variant], [Count as Variant])

Removes the last redo actions that can be performed on the surface.

Type	Description
Action as Variant	<p>[optional] A long expression that specifies the action being remove. If missing or -1, all actions are removed.</p> <p>The Action parameter can be one of the following:</p> <ul style="list-style-type: none"><li>• exUndoRedoAddElement(13) ~ <b>"AddElement;ELEMENTID"</b>, indicates that a new element has been created</li><li>• exUndoRedoRemoveElement(14) ~ <b>"RemoveElement;ELEMENTID"</b>, indicates that an element has been removed</li><li>• exUndoRedoMoveElement(15) ~ <b>"MoveElement;ELEMENTID"</b>, indicates that an element has been moved or resized</li><li>• exUndoRedoUpdateElement(16) ~ <b>"UpdateElement;ELEMENTID"</b>, indicates that one or more properties of the element has been updated, using the <a href="#">StartUpdateElement</a> / <a href="#">EndUpdateElement</a> methods</li><li>• exUndoRedoAddLink(10) ~ <b>"AddLink;LINKID"</b>, indicates that a new link has been created</li><li>• exUndoRedoRemoveLink(11) ~ <b>"RemoveLink;LINKID"</b>, indicates that a link has been removed</li><li>• exUndoRedoUpdateLink(12) ~ <b>"UpdateLink;LINKID"</b>, specifies that one of more properties of the link has been updated, using the <a href="#">StartUpdateLink</a> / <a href="#">EndUpdateLink</a> methods</li></ul> <p>For instance, RedoRemoveAction(12) removes only AddElement actions from the redo stack.</p>
	<p>[optional] A long expression that indicates the number of actions to remove. If missing or -1, all actions are removed. For instance, RedoRemoveAction(12,1) removes only the last AddElement action from the redo stack</p>

The `RedoRemoveAction` method removes the first action to be performed if the `Redo` method is invoked. Use the `RedoRemoveAction()` ( with no parameters ) to remove all redo actions. Use the [UndoRemoveAction](#) method to remove the last action from the undo queue. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- **"EndBlock"**, specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

# method Surface.Refresh ()

Refreshes the control.

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

# method Surface.RemoveSelection ()

Removes the elements in the selection.

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

# method Surface.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 user can add images at design time, by drag and drop files to control's images holder. The [ShowImageList](#) property available for the /COM shows or hides the control's images holder at design mode. Use the [Pictures](#) / [ExtraPictures](#) properties to display different pictures on the element.

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

```
i = ExSurface1.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 = ExSurface1.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:

```
ExSurface1.Replacelcon 0, i, where i specifies the index of icon removed.
```

The following VB clears the control's icons collection:

```
ExSurface1.Replacelcon 0, -1
```



# method Surface.SaveXML (Destination as Variant)

Saves the control's content as XML document to the specified location, using the MSXML parser.

Type	Description
	<p>This object can represent a file name, an XML document object, or a custom object that supports persistence as follows:</p> <ul style="list-style-type: none"><li>String - Specifies the file name. Note that this must be a file name, rather than a URL. The file is created if necessary and the contents are entirely replaced with the contents of the saved document. For example:  <code>Surface1.SaveXML("sample.xml")</code></li><li>Reference to a String member - Saves the control's content to the string member. Note that the string member must be empty, before calling the SaveXML method. For example:  <code>Dim s As String Surface1.SaveXML s</code>  In VB.NET for /NET assembly, you should call such as :  <code>Dim s As String = String.Empty Exsurface1.SaveXML(s)</code>  In C# for /NET assembly, you should call such as :  <code>string s = string.Empty; exsurface1.SaveXML(ref s);</code></li><li>XML Document Object. For example:  <code>Dim xmldoc as Object Set xmldoc = CreateObject("MSXML.DOMDocument") Surface1.SaveXML(xmldoc)</code></li><li>Custom object supporting persistence - Any other custom COM object that supports <b>QueryInterface</b> for <b>IStream</b>, <b>IPersistStream</b>, or <b>IPersistStreamInit</b> can also be provided here and the document will be saved accordingly. In the <b>IStream</b> case, the <b>IStream::Write</b></li></ul>

method will be called as it saves the document; in the **IPersistStream** case, **IPersistStream::Load** will be called with an **IStream** that supports the **Read**, **Seek**, and **Stat** methods.

Return	Description
Boolean	A Boolean expression that specifies whether saving the XML document was ok.

The SaveXML method uses the MSXML ( MSXML.DOMDocument, XML DOM Document ) parser to save the control's data in XML documents. The [LoadXML](#) method loads XML documents being created with SaveXML method. The elements of the surface go to the **<Elements>** section, while the links go to the **<Links>** section. Each element goes under the **<element>** section, while each information about a link goes to the **<link>** section.

The [XML Format](#) of the file is:

```
- <Content Author Component Version ...>
  - <Elements>
    <Element ID ... />
    ...
  </Elements>
  - <Links>
    <Link ID ...>
    </Link>
  </Links>
</Content>
```

# property Surface.ScrollPos(Vertical as Boolean) as Long

Specifies the vertical/horizontal scroll position.

Type	Description
Vertical as Boolean	A Boolean expression that specifies the surface's vertical or horizontal positiuion
Long	A long expression that specifies the scroll position.

The control's ScrollPos, [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position. The ScrollX property is equivalent with ScrollPos(False), and the ScrollY property is equivalent with the ScrollPos(True). Use the [ScrollTo](#) method of the control to scroll the surface at specified position. The [MovePoint](#) method of the control moves the surface from the one point to another. The [MoveCorner](#) method scrolls the surface from a corner to another. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The [AllowMoveSurface](#) property specifies the combination of keys that allows the user to move the surface.

# method Surface.ScrollTo (X as Long, Y as Long)

Scrolls the surface to giving position.

Type	Description
X as Long	A Long expression that specifies the x-position where the surface should be scrolled.
Y as Long	A Long expression that specifies the y-position where the surface should be scrolled.

Use the ScrollTo method of the control to scroll the surface at specified position. The [MovePoint](#) method of the control moves the surface from the one point to another. The [MoveCorner](#) method scrolls the surface from a corner to another. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position.

# property Surface.ScrollX as Long

Indicates the x-scrolling position of the surface.

Type	Description
Long	A Long expression that specifies the x-scroll position.

The control's [ScrollPos](#), ScrollX and [ScrollY](#) properties specify the surface's scroll position. The ScrollX property is equivalent with ScrollPos(False), and the ScrollY property is equivalent with the ScrollPos(True). Use the [ScrollTo](#) method of the control to scroll the surface at specified position. The [MovePoint](#) method of the control moves the surface from the one point to another. The [MoveCorner](#) method scrolls the surface from a corner to another. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The [AllowMoveSurface](#) property specifies the combination of keys that allows the user to move the surface.

# property Surface.ScrollY as Long

Indicates the y-scrolling position of the surface.

Type	Description
Long	A Long expression that specifies the y-scroll position.

The control's [ScrollPos](#), [ScrollX](#) and [ScrollY](#) properties specify the surface's scroll position. The [ScrollX](#) property is equivalent with [ScrollPos\(False\)](#), and the [ScrollY](#) property is equivalent with the [ScrollPos\(True\)](#). Use the [ScrollTo](#) method of the control to scroll the surface at specified position. The [MovePoint](#) method of the control moves the surface from the one point to another. The [MoveCorner](#) method scrolls the surface from a corner to another. The [ScrollTo](#) method ensures that the element fits the surface's visible area. The [AllowMoveSurface](#) property specifies the combination of keys that allows the user to move the surface.

# property Surface.SelCount as Long

Indicates the number of elements being selected on the surface.

Type	Description
Long	A Long expression that specifies the number of selected elements.

The SelCount property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [Selection](#) property sets or gets a safe array of selected elements. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.

# method Surface.SelectAll ()

Selects all selectable elements in the control.

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

The SelectAll method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.



# property Surface.Selection as Variant

Returns or sets a safe array of selected elements on the surface.

Type	Description
Variant	A Safe-Array of <a href="#">Element</a> objects. If calling the Set property the Safe-Array may contains the ID of the Elements to be selected, like: Selection = Array("1", "2"), which indicates that the elements with the ID, "1" and "2" are selected. The Selection method can be used to enumerate the selected elements using the for each statements.

The [Selection](#) property sets or gets a safe array of selected elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.

# Property Surface.SelectObjectColor as Color

Indicates the color to show the selected objects.

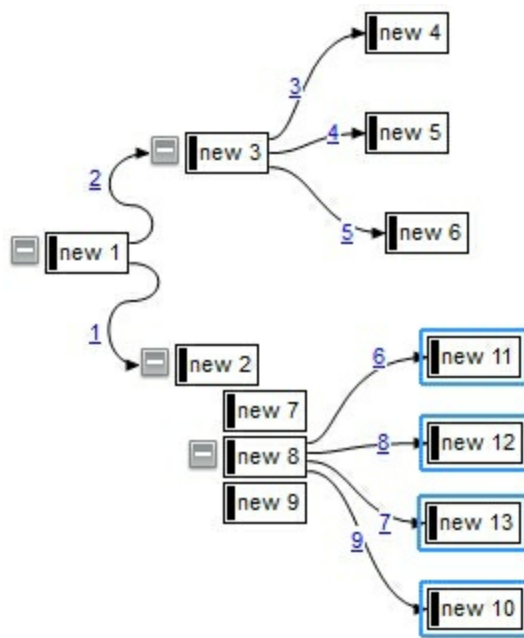
Type	Description
Color	A Color expression that specifies the color to show the selected elements while the control has the focus.

The `SelectObjectColor` / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The [SelectObjectStyle](#) property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background\(exSelectObjectRectColor\)](#) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

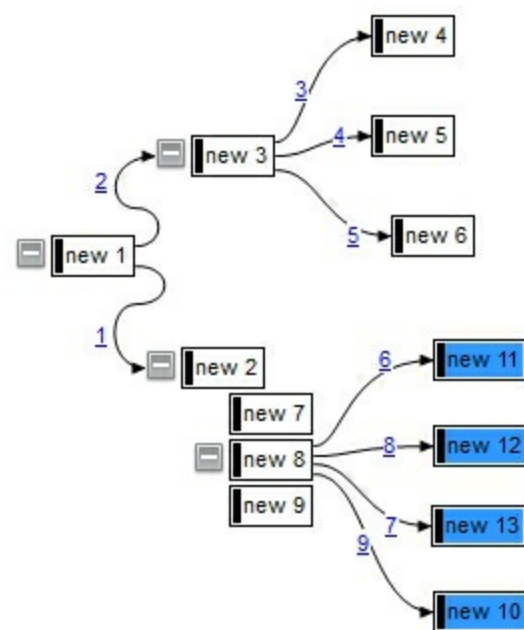
The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the `Element` object indicates whether the element is selected or unselected. The [Selectable](#) property of the `Element` object indicates whether the element is selectable or un-selectable.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

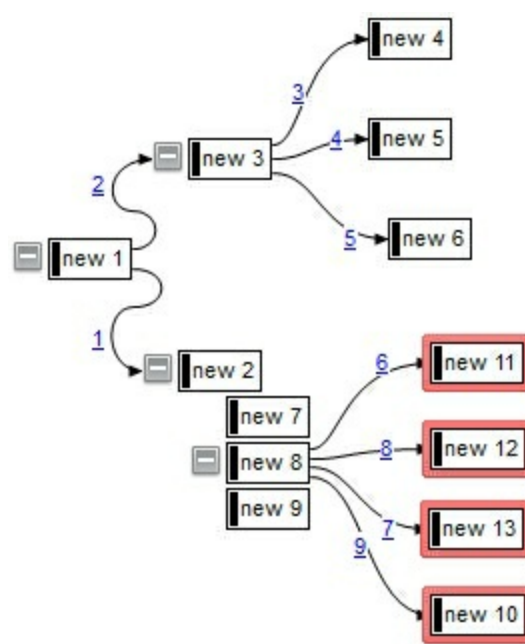
The following screen shot shows the selected elements ( default, `SelectObjectStyle` property `exLinesSolid + exLinesThick`, `SelectObjectColor` indicates a solid color ):



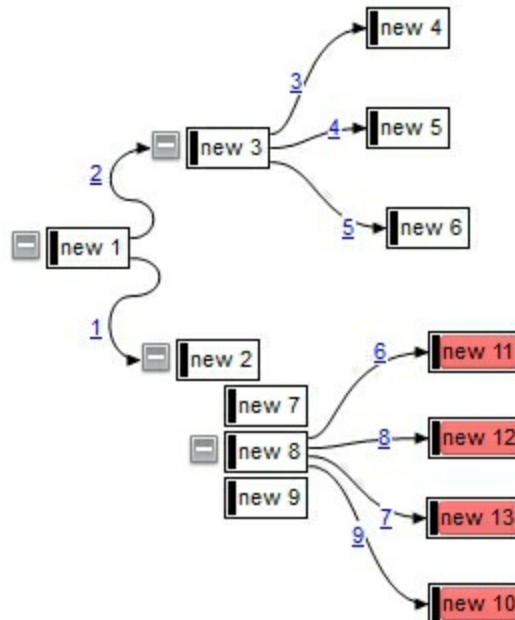
The following screen shot shows the selected elements ( default, SelectObjectStyle property exNoLines, SelectObjectColor indicates a solid color ):



The following screen shot shows the selected elements ( default, SelectObjectStyle property exLinesSolid + exLinesThick, SelectObjectColor indicates an EBN color ):



The following screen shot shows the selected elements ( default, SelectObjectStyle property exNoLines, SelectObjectColor indicates an EBN color ):



# property Surface.SelectObjectColorInactive as Color

Indicates the color to show the selected objects, when the surface is not active/focused.

Type	Description
Color	A Color expression that specifies the color to show the selected elements while the control is not focused.

The `SelectObjectColorInactive` / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectStyle](#) property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background\(exSelectObjectRectColor\)](#) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the `Element` object indicates whether the element is selected or unselected. The [Selectable](#) property of the `Element` object indicates whether the element is selectable or un-selectable.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

# property Surface.SelectObjectStyle as LinesStyleEnum

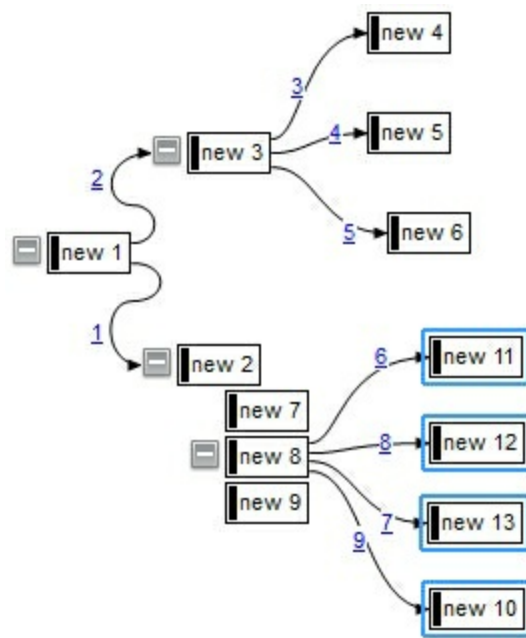
Specifies the style to display the selected object.

Type	Description
<a href="#">LinesStyleEnum</a>	A LinesStyleEnum expression that specifies the style of the line to be shown on the selected elements.

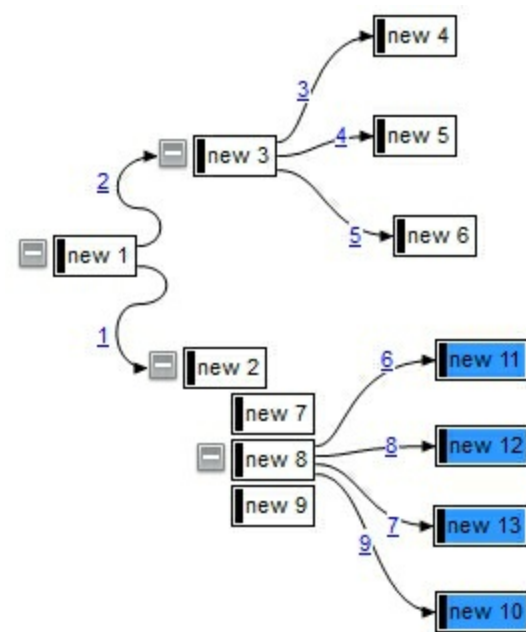
By default, the The SelectObjectStyle property is exLinesSolid + exLinesThick. If the SelectObjectStyle property is exNoLines, no lines are shown around the selected elements. The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

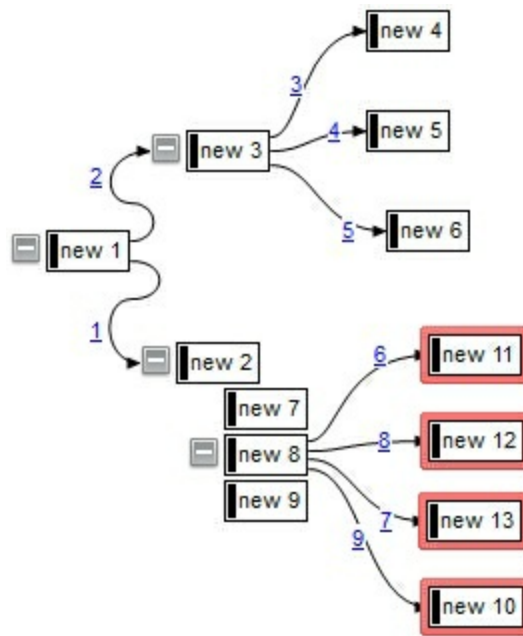
The following screen shot shows the selected elements ( default, SelectObjectStyle property exLinesSolid + exLinesThick, SelectObjectColor indicates a solid color ):



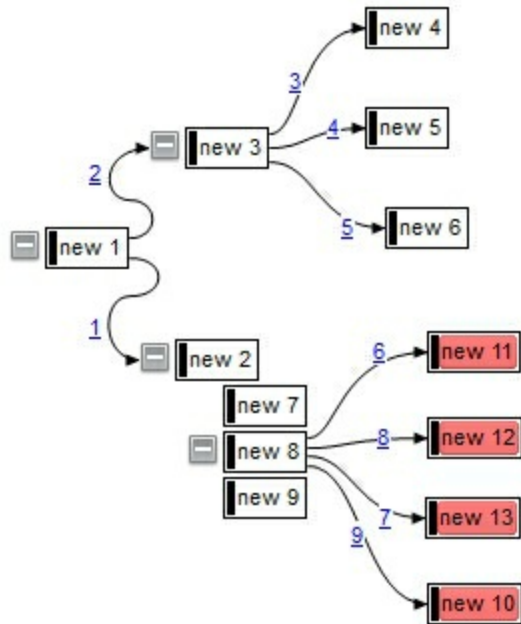
The following screen shot shows the selected elements ( default, SelectObjectStyle property exNoLines, SelectObjectColor indicates a solid color ):



The following screen shot shows the selected elements ( default, SelectObjectStyle property exLinesSolid + exLinesThick, SelectObjectColor indicates an EBN color ):



The following screen shot shows the selected elements ( default, SelectObjectStyle property exNoLines, SelectObjectColor indicates an EBN color ):





# property Surface.SelectObjectTextColor as Color

Indicates the color to show the text for selected objects.

Type	Description
Color	A Color expression that specifies the foreground color for selected elements. <i>The -1 indicates that no foreground is changed for selected elements.</i>

By default, the SelectObjectTextColor property is -1, which indicates no foreground color is changed for selected elements. The [SelectObjectColor](#) / SelectObjectTextColor property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The [SelectObjectStyle](#) property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

# property Surface.SelectObjectTextColorInactive as Color

Indicates the color to show the text for selected objects, when the surface is not active/focused.

Type	Description
Color	A Color expression that specifies the foreground color for selected elements. <i>The -1 indicates that no foreground is changed for selected elements.</i>

By default, the SelectObjectTextColorInactive property is -1, which indicates no foreground color is changed for selected elements. The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectStyle](#) property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

# property Surface.SelElement (Index as Long) as Element

Gets the element being selected giving its index in the selection.

Type	Description
Index as Long	A Long expression that specifies the index of selected element to be accessed. The Index is a positive number between 0 and ( <a href="#">SelCount</a> - 1) property
<a href="#">Element</a>	An Element object that specifies the selected element

The SelElement property returns the selected element based on its index in the selected elements collection. The [SelCount](#) property counts the number of selected elements. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. The [Selection](#) property sets or gets a safe array of selected elements. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.

# property Surface.ShowGridLines as Boolean

Shows or hides the grid lines in the control.

Type	Description
Boolean	A Boolean expression that specifies whether the minor/major grid lines are shown or hidden.

By default, the ShowGridLines property is True. Use the ShowGridLines property to specify whether the control shows or hides the minor/major grid lines. Use the [MinorGridWidth](#) / [MinorGridHeight](#) property to specify the how minor grid lines are displayed/aligned. Use the [MajorGridWidth](#) / [MajorGridHeight](#) property to specify the how major grid lines are displayed/aligned. Use the [MajorGridStyle](#) property to specify the style of the major lines. Use the [MinorGridStyle](#) property to specify the style of the minor lines. The [MajorGridColor](#) property specifies the color to show the major grid lines. The [MinorGridColor](#) property specifies the color to show the minor grid lines. Use the [AxisStyle](#) property to hide the axis lines or to display with a different style. Use the [AxisColor](#) property to specify the color to show the axis lines.

Use the [AlignObjectsToGridLines](#) property to align the elements to the grid lines. The [AutoSize](#) property of the Element specifies whether the element's size is computed based on the element's content. The [CaptionAlign](#) property specifies the alignment of the element's caption.

# property Surface.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.

The property is available for /COM version only, and only at design mode. By default, the ShowImageList property is False. Use the ShowImageList property to show 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 control. Use the [Replacelcon](#) method to add, remove or clear icons in the control's images collection, at runtime. Use the [Pictures](#) / [ExtraPictures](#) properties to display different pictures on the element.

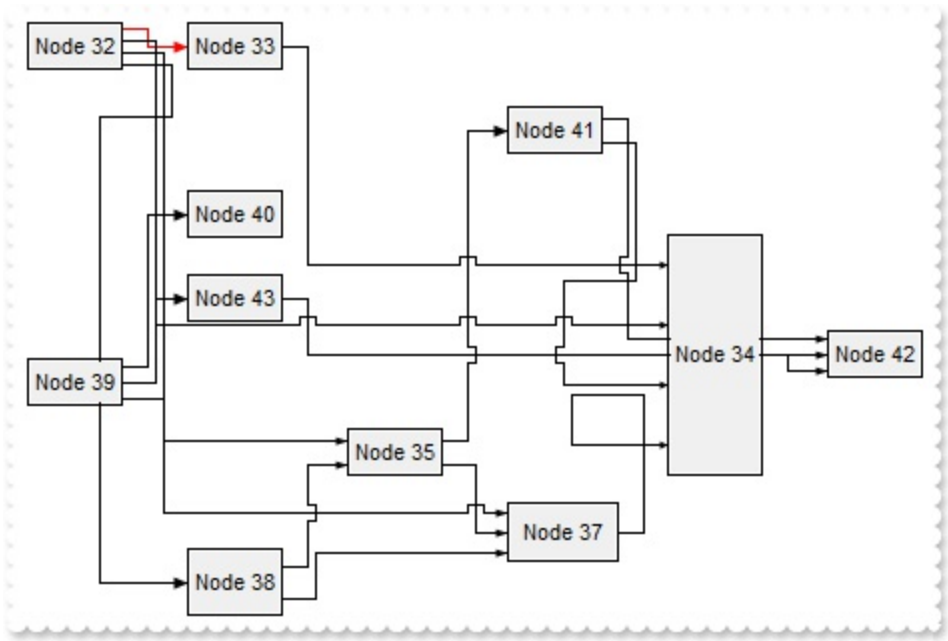
# property Surface.ShowLinks as ShowExtendedLinksEnum

Retrieves or sets a value that indicates whether the links between elements are visible or hidden.

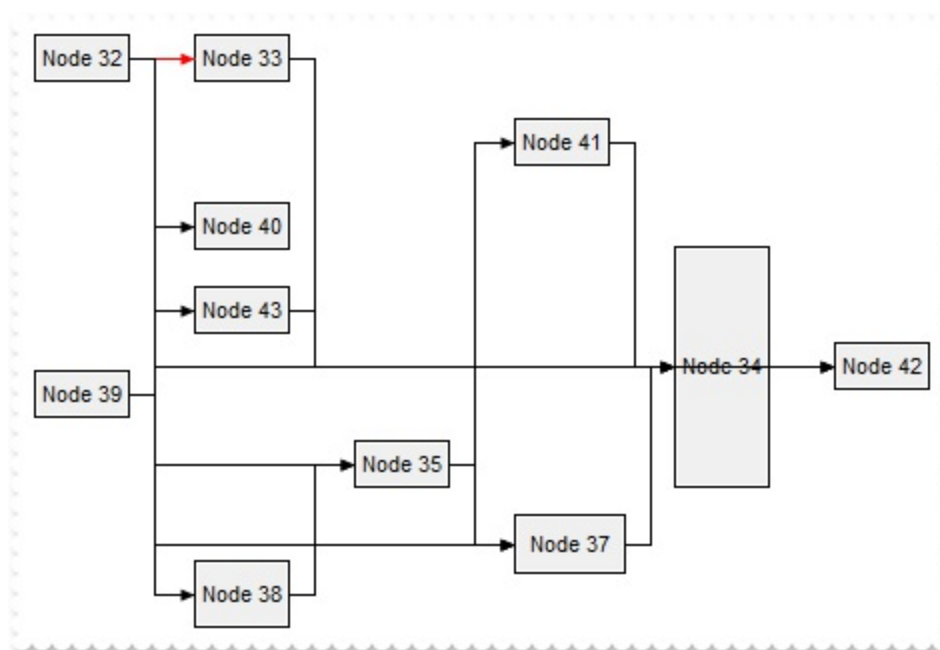
Type	Description
<a href="#">ShowExtendedLinksEnum</a>	A ShowExtendedLinksEnum expression that specifies whether the links on the surface are shown or hidden.

By default, the ShowLinks property is exShowExtendedLinks + exShowLinksFront. The ShowLinks property specifies the way the control shows the links on the surface. Use the ShowLinks property to hide the links on the surface. Use the ShowLinks property to show the links on the control's background rather than front. Use the [Visible](#) property of the Link to hide a specific link. Use the [ShowLinksOnCollapse](#) property to show the links between an element and collapsed elements. The [ExpandLinkedElements](#) property specifies whether the elements displays the expand/collapse glyphs when the element has outgoing elements ( the [OutgoingLinks](#) property specifies the links that starts from the element ).

The following screen shot shows the links on the surface ( when the ShowLinks property is exShowExtendedLinks + exShowCrossLinksRect ):



The following screen shot shows the links on the surface ( when the ShowLinks property is exShowLinks ):



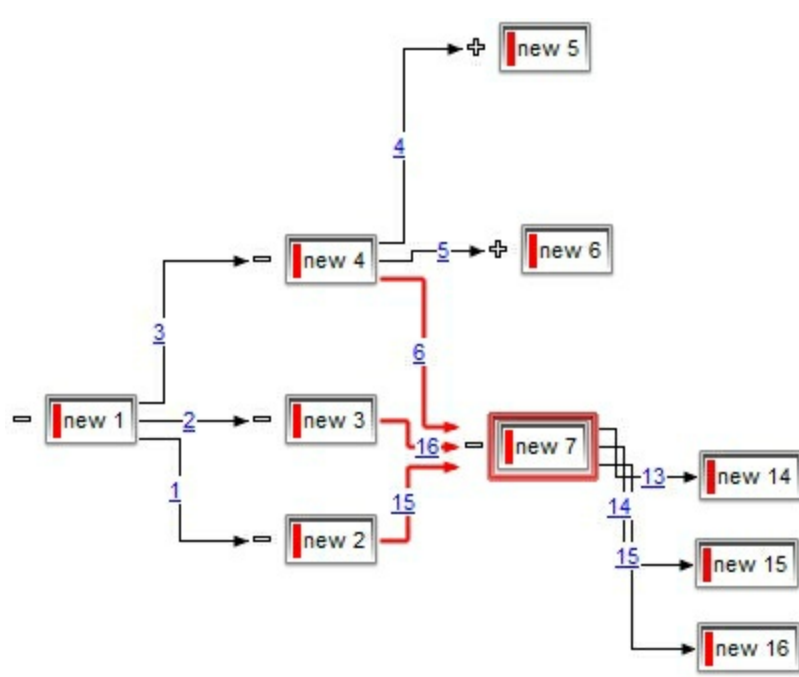
# property Surface.ShowLinksColor(Links as ShowLinksEnum) as Color

Retrieves or sets a value that indicates the color to display the links based on the user selection.

Type	Description
Links as <a href="#">ShowLinksEnum</a>	A ShowLinksEnum expression that indicates the color of incoming, outgoing or collapsed links is changed.
Color	A Color expression that specifies the color to show the related links.

The ShowLinksColor property specifies a color to show the incoming, outgoing or collapsed links. The [Color](#) property specifies the color to show the entire link. For instance, Use the [ShowLinksColor](#)(exShowLinksStartsFrom) / [ShowLinksStyle](#)(exShowLinksStartsFrom) / [ShowLinksWidth](#)(exShowLinksStartsFrom) properties to mark the outgoing links of selected elements. The control fires the [SelectionChanged](#) event when a new element is selected or unselected.

The following screen shot shows the incoming links (red ):





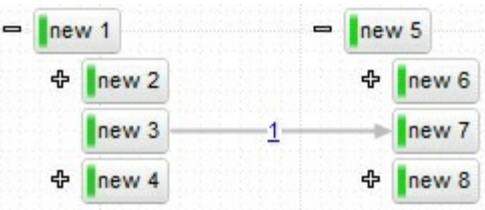
# property Surface.ShowLinksOnCollapse as Boolean

Specifies whether the links for collapsed elements are shown or hidden.

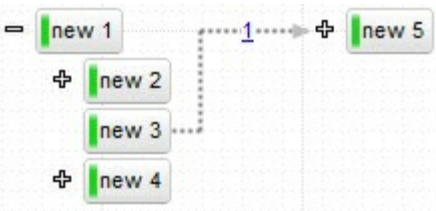
Type	Description
Boolean	A Boolean expression that specifies whether the links of collapsed elements are shown or hidden.

By default, ShowLinksOnCollapse property is True. Use the ShowLinksOnCollapse property to show the links between collapsed elements. Use the [Expanded](#) property to specify whether the element is expanded or collapsed. The [ExpandLinkedElements](#) property specifies whether the elements displays the expand/collapse glyphs when the element has outgoing elements ( the [OutgoingLinks](#) property specifies the links that starts from the element ). Use the [Visible](#) property of the Link to hide a specific link. The [ShowLinksColor](#)(exShowCollapsedLinks) / [ShowLinksStyle](#)( exShowCollapsedLinks) / [ShowLinksWidth](#)(exShowCollapsedLinks) property specifies how the collapsed links are shown.

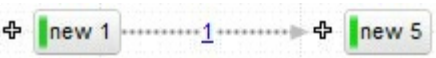
The following screen shot shows a link between two child elements, when both parents are expanded:



Having the ShowLinksOnCollapse property on True, the following screen shot shows the link when one parent is collapsed:



Having the ShowLinksOnCollapse property on True, the following screen shot shows the link when both parents are collapsed:



In other words, the ShowLinksOnCollapse property allows you to still display the links for collapsed elements. A collapsed link is shown between expanded and visible elements.

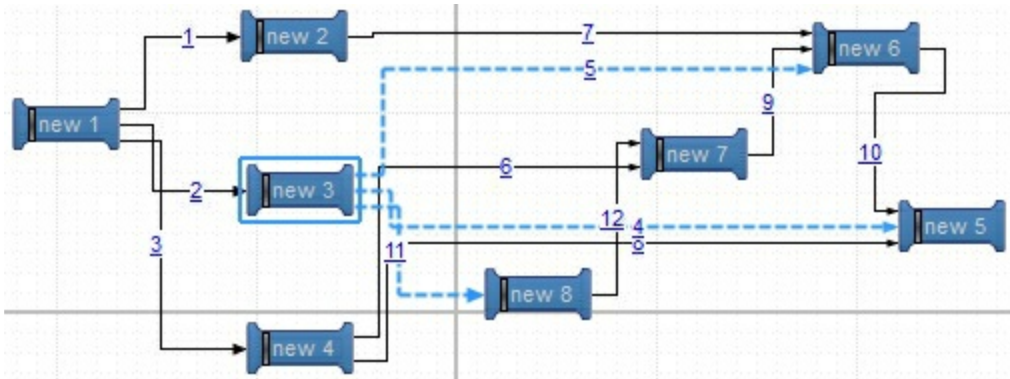
# property Surface.ShowLinksStyle(Links as ShowLinksEnum) as LinkStyleEnum

Retrieves or sets a value that indicates the style to display the links based on the user selection.

Type	Description
Links as <a href="#">ShowLinksEnum</a>	A ShowLinksEnum expression that indicates the width of incoming, outgoing or collapsed links is changed.
<a href="#">LinkStyleEnum</a>	A LinkStyleEnum expression that specifies the style of the line to show the related links.

The ShowLinksStyle property specifies the style to show the incoming, outgoing or collapsed links. The [Style](#) property indicates the style of the line to be shown on a particular link. Use the [ShowLinksColor](#)(exShowLinksStartsFrom) / [ShowLinksStyle](#)(exShowLinksStartsFrom) / [ShowLinksWidth](#)(exShowLinksStartsFrom) properties to mark the outgoing links of selected elements. The control fires the [SelectionChanged](#) event when a new element is selected or unselected. The [OutgoingLinks](#) property specifies the list of links that starts from the current element. The [IncomingLinks](#) property specifies the list of links that ends on the current element.

The following screen shot shows the outgoing links with a different style:



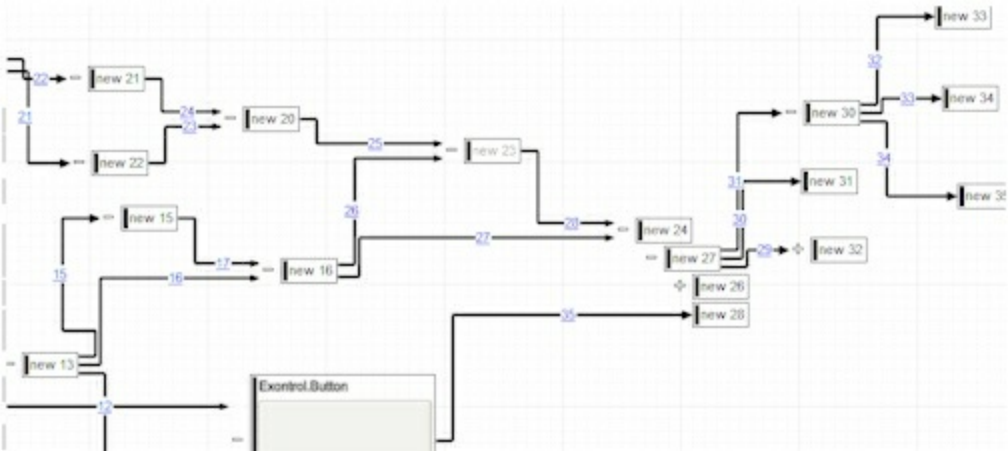
# property Surface.ShowLinksType as ShowLinkTypeEnum

Specifies how the links are displayed between the elements.

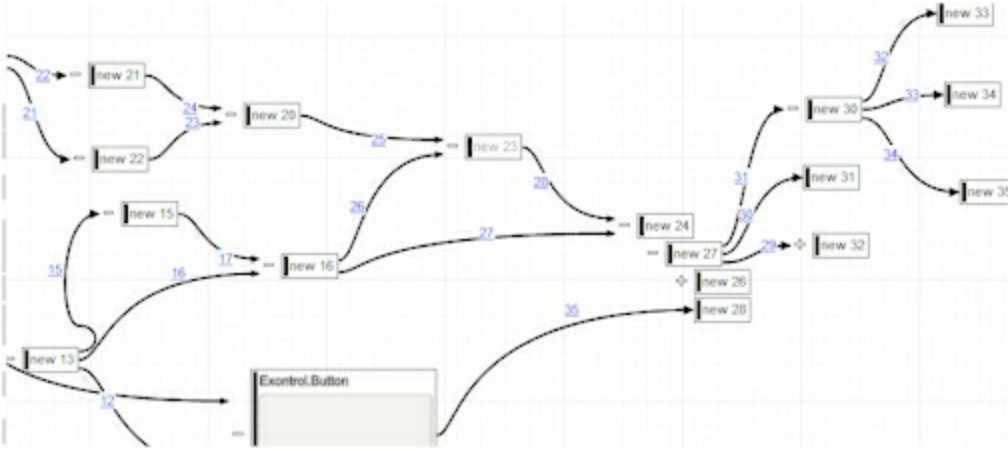
Type	Description
<a href="#">ShowLinkTypeEnum</a>	A ShowLinkTypeEnum expression that defines the type of the link to be show between elements.

The ShowLinksType property defines the type of the link to be shown between elements on the surface. The [ShowLinkType](#) property specifies a different type of link between two elements. Use the [LinksColor](#) property to define the color to show all links on the surface. The control's [LinksShowDir](#) property specifies whether the arrow of the links is shown or hidden. The [LinksWidth](#) property specifies the size of the links and so the size of the arrow. The [ShowLinks](#) property specifies whether the surface shows or hides the links. The control's [LinksStyle](#) property defines the style of the line to be shown on the link.

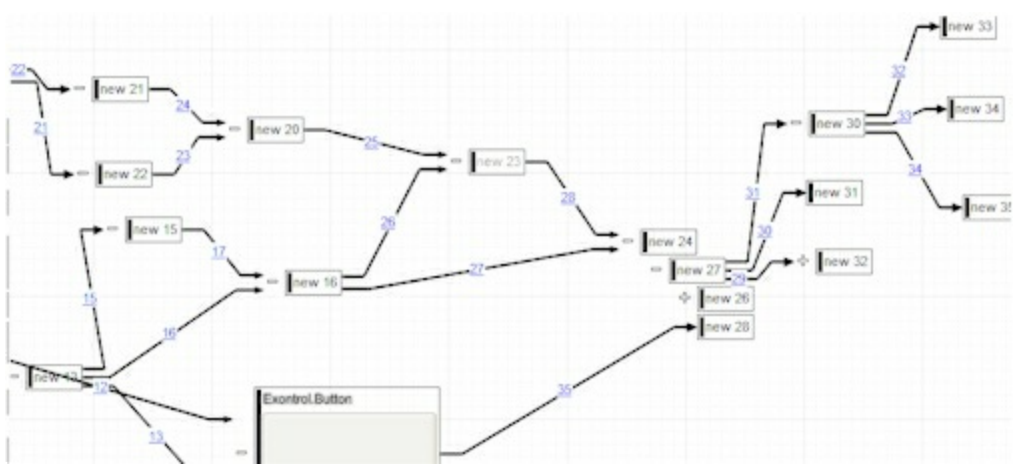
The following screen shot shows the exLinkRectangular type:



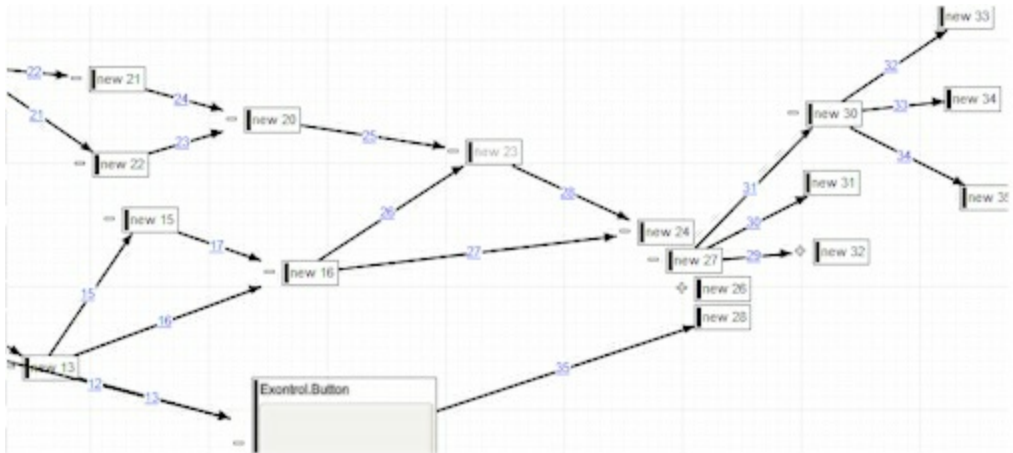
The following screen shot shows the exLinkRound type:



The following screen shot shows the exLinkDirect type:



The following screen shot shows the exLinkStraight type:



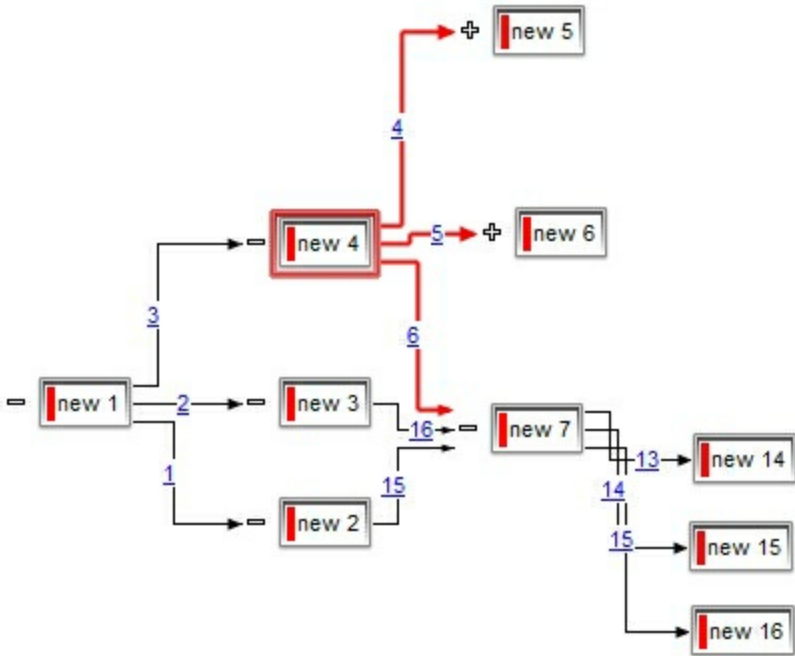
# property Surface.ShowLinksWidth(Links as ShowLinksEnum) as Long

Retrieves or sets a value that indicates the width to display the links based on the user selection.

Type	Description
Links as <a href="#">ShowLinksEnum</a>	A ShowLinksEnum expression that indicates the width of incoming, outgoing or collapsed links is changed.
Long	A Long expression that specifies the width of related links.

The ShowLinksWidth property specifies the width to show the incoming, outgoing or collapsed links. The [Width](#) property specifies the width of a particular link. Use the [ShowLinksColor](#)(exShowLinksStartsFrom) / [ShowLinksStyle](#)(exShowLinksStartsFrom) / [ShowLinksWidth](#)(exShowLinksStartsFrom) properties to mark the outgoing links of selected elements. The control fires the [SelectionChanged](#) event when a new element is selected or unselected. The [OutgoingLinks](#) property specifies the list of links that starts from the current element. The [IncomingLinks](#) property specifies the list of links that ends on the current element.

The following screen shot shows the outgoing links (red ):



**method Surface.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.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using `<a ;exp=>` or `<a ;e64=>` anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu</a>`" that displays `show lines-` in gray when the user clicks the `+` anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY" string encodes the "`<fgcolor 808080>show lines<a>-</a></fgcolor>`" The `Decode64Text/Encode64Text` methods of the `eXPrint` can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "`<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3`" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the `+` sign.



- **<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&gt;**;

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the <font face;size> to define a smaller or a larger font to be displayed. For instance: "Text with <font ;7><off 6>subscript" displays the text such as: Text with subscript The "Text with <font ;7><off -6>superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The <font> HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<font ;18><gra FFFFFFFF;1;1>gradient-center</gra></font>" generates the following picture:

gradient-center

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

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

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

outline anti-aliasing

# property Surface.SingleSel as Boolean

Returns or sets a value that indicates whether the user can select one or more objects.

Type	Description
Boolean	A Boolean expression that specifies whether the control supports single or multiple selection.

By default, the SingleSel property is False, which indicates that multiple elements can be selected. The SingleSel property specifies whether the surface allows selecting one or multiple elements. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface. Use the [AllowMoveSelection](#) property to prevent moving the entire selection when focused element is moved. Use the [AllowResizeSelection](#) property to prevent resizing the entire selection when focused element is resized. Use the [AllowMoveDescendents](#) property to prevent moving all descendents ( children and outgoing elements ) when focused element is moved. The [HideSel](#) property specifies whether the selected elements are highlighted or not when the control loses the focus.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.

# method Surface.StartBlockUndoRedo ()

Starts recording the UI operations as a block of undo/redo operations.

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

You can use the StartBlockUndoRedo / [EndBlockUndoRedo](#) methods to group multiple Undo/Redo operations into a single-block. The GroupUndoRedoActions groups the next to current Undo/Redo Actions in a single block. A block may hold multiple Undo/Redo actions. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. Use the GroupUndoRedoActions method to group two or more entries in the Undo/Redo queue in a single block, so when a next Undo/Redo operation is performed, multiple actions may occur. For instance, moving several elements in the same time ( multiple elements selection ) is already recorded as a single block. Use the [UndoRedoQueueLength](#) property to specify the number of entries that Undo/Redo queue may store.

A block starts with StartBlock and ends with EndBlock when listed by [UndoListAction](#)/[RedoListAction](#) property as in the following sample:

```
StartBlock
MoveElement;B
MoveElement;A
EndBlock
```

# property Surface.Template as String

Specifies the control's template.

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

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

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

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

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

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

An instruction can be one of the following:

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

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

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

The Template supports the following general functions:

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

# property Surface.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 Surface.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 Surface.ToolBarCaption(ID as Long) as String

Specifies the HTML caption of the giving ToolBar in the control's toolbar.

Type	Description
ID as Long	<p>A Long expression that indicates the identifier of the caption/field being changed. The predefined identifiers are:</p> <ul style="list-style-type: none"><li>• <b>100</b>, Home (restores the view to the origin)</li><li>• <b>101</b>, Zoom (changes the zooming factor of the surface)</li><li>• <b>103</b>, Undo ( undoes the last control operation, enabled only if the Undo operation is possible)</li><li>• <b>104</b>, Redo ( redoes the next action in the control's redo queue, enabled only if the Undo operation is possible)</li></ul>
String	<p>A string expression that indicates the caption or field's value.</p>

Use the ToolBarCaption property to change captions in the control's toolbar. The [ToolBarToolTip](#) property specifies the button's tooltip. The [ToolBarImages](#) method loads icons to be displayed on the control's toolbar. The [ToolBarHTMLPicture](#) property loads custom-sized pictures to be displayed on the control's toolbar using the <img> HTML tag. Use the [ToolBarFormat](#) property to customize the control's toolbar such as adding new buttons, icons, pictures or HTML captions to control's toolbar. The control fires the [ToolBarClick](#) event when the user clicks a button in the control's toolbar. For exToolBar... values ( any value greater that 100 ), # character splits the button's label and it's identifier ( SelectedID parameter ). For instance, if the ToolBarCaption(200) = "Letter#1", the button with the identifier 200 displays the "Letter" label, while the 1 is carried to SelectedID parameter of the [ToolBarClick](#) event when it is fired. If the ToolBarCaption property includes vbCrLf ( "\r\n" sequence ), the associated buttons displays a drop down field. Use the [ToolBarRefresh](#) method to refresh the control's toolbar.

The ToolBarCaption property supports the following HTML elements:

- **<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.

The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using `<a ;exp=>` or `<a ;e64=>` anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu</a>`" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY" string encodes the "`<fgcolor 808080>show lines<a>-</a></fgcolor>`" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "`<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3`" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **<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=rrgbb> ... </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=rrgbb> ... </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=rrgbb> ... </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=rrgbb> ... </dotline>` draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ...`

`</dotline>` draws a black dot-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.

- **`<upline> ... </upline>`** draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- **`<r>`** right aligns the text
- **`<c>`** centers the text
- **`<br>`** forces a line-break
- **`<img>number[:width]</img>`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`<img>key[:width]</img>`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **`&`** glyph characters as **`&amp;`**; ( `&` ), **`&lt;`**; ( `<` ), **`&gt;`**; ( `>` ), **`&qout;`**; ( `"` ) and **`&#number;`**; ( the character with specified code ), For instance, the `&#8364;` displays the EUR character. The `&` ampersand is only recognized as markup when it is followed by a known letter or a `#`character and a digit. For instance if you want to display `<b>bold</b>` in HTML caption you can use `&lt;b&gt;bold&lt;/b&gt;`;
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `<font face;size>` to define a smaller or a larger font to be displayed. For instance: "Text with `<font ;7><off 6>`subscript" displays the text such as: Text with subscript The "Text with `<font ;7><off -6>`superscript" displays the text such as: Text with subscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the `rr/gg/bb` represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `<font>` HTML tag can be used to define the height of the font. Any of the `rrggbb`, `mode` or `blend` field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "`<font ;18><gra FFFFFFFF;1;1>`gradient-center`</gra></font>`" generates the following picture:

gradient-center

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

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The <font> HTML tag can be used to define the height of the font. For instance the "<font ;31><sha>shadow</sha></font>" generates the following picture:

shadow

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

outline anti-aliasing



# property Surface.ToolBarFormat as String

Specifies the CRD format to arrange the buttons inside the control's toolbar.

Type	Description
String	A String expression that specifies the <a href="#">CRD</a> format to control's toolbar.

By default, the ToolBarFormat property is "-1,100,101" and arranges the toolbar as in the following screen shot. If empty, the control's toolbar displays no buttons.



The predefined identifiers for ToolBarFormat property are:

- **100**, Home (restores the view to the origin)
- **101**, Zoom (changes the zooming factor of the surface)
- **103**, Undo (undoes the last control operation, enabled only if the Undo operation is possible)
- **104**, Redo (redoes the next action in the control's redo queue, enabled only if the Undo operation is possible)

Use the ToolBarFormat property to add new buttons, to display icons, pictures, or any other HTML caption. The [ToolBarCaption](#) property specifies the caption of the button. The [ToolBarToolTip](#) property specifies the button's tooltip. The control fires the [ToolBarClick](#) event when the user clicks a button in the control's toolbar. The control fires the [ToolBarAnchorClick](#) event when the user clicks an hyperlink element. The [ToolBarRefresh](#) method refreshes the control's toolbar.

For instance, the following screen shot shows the control's toolbar when the ToolBarFormat property is "-1,100,101,1000"



If the [ToolBarCaption](#)(1000) property is set as "<sha ;;0>custom" the control's toolbar shows as:



# property Surface.ToolBarHTMLPicture(Key as String) as Variant

Adds or replaces a picture in toolbar's HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none"><li>• a string expression that indicates the path to the picture file, being loaded.</li><li>• a string expression that indicates the base64 encoded string that holds a picture object, Use the <a href="#">eximages</a> tool to save your picture as base64 encoded format.</li><li>• A Picture object that indicates the picture being added or replaced. ( A Picture object implements IPicture interface ),</li></ul> <p>If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added.</p>

By default, the ToolBarHTMLPicture collection is empty. The ToolBarHTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the <img> tags. Use the ToolBarHTMLPicture property to add new pictures to be used in HTML captions. For instance, the ToolBarHTMLPicture("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 ToolBarHTMLPicture property retrieves a Picture object ( this implements the IPictureDisp interface ). The [ToolBarImages](#) method specifies the list of 16x16 icons to be displayed on the control's toolbar. The [ToolBarCaption](#) property specifies the caption of the button ( including icons, picture and so on ).

## method Surface.ToolBarImages (Handle as Variant)

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

Type	Description
Handle as Variant	A long expression that identifies a handle to an Image list ( the Handle should be of HIMAGELIST type ) or a string expression that indicates the base64 encoded string that holds the icons list. Use the <a href="#">eximages</a> tool to save your icons as base64 encoded format.

The ToolBarImages method assigns a list of icons to be displayed on the control's toolbar. The icons can be displayed on the control's toolbar using the <img>number</img> HTML tags. The [ToolBarHTMLPicture](#) property handles a collection of custom size picture being displayed in the HTML captions, using the <img>key</img> tags. The [ToolBarReplacelcon](#) method replaces icons in the control's toolbar. The [ToolBarCaption](#) property specifies the caption of the button ( including icons, picture and so on ).

# method Surface.ToolBarRefresh ()

Refreshes the control's toolbar.

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

Use the ToolBarRefresh method to refresh the control's toolbar. The [ToolBarCaption](#) property specifies the caption of the button ( including icons, picture and so on ).

# method Surface.ToolBarReplacelcon ([Icon as Variant], [Index as Variant])

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

Type	Description
Icon as Variant	A long expression that indicates the icon's handle. By default, the Icon parameter is 0, if it is missing.
Index as Variant	A long expression that indicates the index where icon is inserted. By default, the Index parameter is -1, if it is missing.
Return	Description
Long	A long expression that indicates the index of the icon in the images collection

Use the ToolBarReplacelcon property to add, remove or replace an icon in the toolbar's images collection. Also, the ToolBarReplacelcon property can clear the toolbar's images collection. Use the [ToolBarImages](#) method to attach an image list to the toolbar.

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

i = Surface1.ToolBarReplacelcon( LoadPicture("d:\icons\help.ico").Handle), where i is the index to insert the icon

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

i = Surface1.ToolBarReplacelcon( LoadPicture("d:\icons\help.ico").Handle, 0), in this case the i is zero, because the first icon was replaced.

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

Surface1.ToolBarReplacelcon 0, i, in this case the i is the index of the icon to remove

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

Surface1.ToolBarReplacelcon 0, -1

# property Surface.ToolBarToolTip(ID as Long) as String

Specifies the HTML tooltip of the giving item in the control's toolbar.

Type	Description
ID as Long	<p>A Long expression that indicates the identifier of the tooltip/field being changed. The predefined identifiers are:</p> <ul style="list-style-type: none"><li>• <b>100</b>, Home (restores the view to the origin)</li><li>• <b>101</b>, Zoom (changes the zooming factor of the surface)</li><li>• <b>103</b>, Undo ( undoes the last control operation, enabled only if the Undo operation is possible)</li><li>• <b>104</b>, Redo ( redoes the next action in the control's redo queue, enabled only if the Undo operation is possible)</li></ul>
String	<p>A string expression that indicates the HTML tooltip of the field in the control's toolbar.</p>

The ToolBarToolTip property assigns a tooltip to be displayed when mouse-pointer hovers a field in the control's toolbar. The [ToolBarImages](#) method loads icons to be displayed on the control's toolbar. The [ToolBarHTMLPicture](#) property loads custom-sized pictures to be displayed on the control's toolbar using the <img> HTML tag. Use the [ToolBarCaption](#) property to change captions in the control's toolbar.

The ToolBarToolTip property supports the following HTML elements:

- **<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.



The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using <a ;exp=> or <a ;e64=> anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption.

- exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
- e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY</a>" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY" string encodes the "<fgcolor 808080>show lines<a>-</a></fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "<solidline><b>Header</b></solidline><br>Line1<r><a ;exp=show lines>+</a><br>Line2<br>Line3" shows the Header in underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

- **<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&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:  

- **<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:  


- **<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 Surface.ToolBarVisible as Boolean

Shows or hides control's toolbar.

Type	Description
Boolean	A Boolean expression that specifies whether the control's toolbar is visible or hidden.

By default, the ToolBarVisible property is True. Use the ToolBarVisible property to hide the control's toolbar. Use the [ToolBarFormat](#) property to customize the control's toolbar, by adding new buttons or drop-down fields to the control's toolbar. The [ToolBarCaption](#) property specifies the HTML caption to be shown on fields of the control's toolbar. The ToolBarToolTip property assigns a tooltip to a field on the control's toolbar. The [ToolBarClick](#) event occurs once the user selects/clicks a field in the control's toolbar. The [ToolBarAnchorClick](#) event notifies your application if an anchor element is clicked on the control's toolbar. The [ToolbarRefresh](#) method refreshes the control's toolbar.

The following screen shot shows the control's default toolbar:



# property Surface.ToolTipDelay as Long

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

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

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to assign a font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. The [ToolTip/ToolTipTitle](#) property of the Element defines the element's tooltip. The [ToolTip](#) property of the Link defines the link's tooltip. Use the [ShowToolTip](#) method to programmatically show a custom tooltip.

# property Surface.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

Type	Description
IFontDisp	A Font object that defines the font to show the control's tooltip. You can use the <font> HTML tag to define a different font for parts of the tooltip.

Use the [ToolTipFont](#) property to assign a font for the control's tooltip. If the [ToolTipDelay](#) or [ToolTipPopDelay](#) property is 0, the control displays no tooltips. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. The [ToolTip/ToolTipTitle](#) property of the Element defines the element's tooltip. The [ToolTip](#) property of the Link defines the link's tooltip. Use the [ShowToolTip](#) method to programmatically show a custom tooltip.

# property Surface.ToolTipPopDelay as Long

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

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

If the [ToolTipDelay](#) or ToolTipPopDelay property is 0, the control displays no tooltips. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to assign a font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. The [ToolTip/ToolTipTitle](#) property of the Element defines the element's tooltip. The [ToolTip](#) property of the Link defines the link's tooltip. Use the [ShowToolTip](#) method to programmatically show a custom tooltip.

# property Surface.ToolTipWidth as Long

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

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

Use the ToolTipWidth property to specify the width of the tooltip window. If the [ToolTipDelay](#) or [ToolTipPopDelay](#) property is 0, the control displays no tooltips. 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 assign a font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. The [ToolTip/ToolTipTitle](#) property of the Element defines the element's tooltip. The [ToolTip](#) property of the Link defines the link's tooltip. Use the [ShowToolTip](#) method to programmatically show a custom tooltip.

# method Surface.Undo ()

Performs the last Undo operation.

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

The Undo method undoes the last control operation. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [CanUndo](#) method indicates whether the control can perform an Undo operation. The [Redo](#) redoes the next action in the control's redo queue. The [UndoRedoQueueLength](#) property gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue, or in other words how many operations the control's Undo/Redo manager may store.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- **"EndBlock"**, specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked.

# property Surface.UndoListAction ([Action as Variant], [Count as Variant]) as String

Lists the Undo actions that can be performed on the surface.

Type	Description
Action as Variant	<p>[optional] A long expression that specifies the action being listed. If missing or -1, all actions are listed.</p> <p>The Action parameter can be one of the following:</p> <ul style="list-style-type: none"><li>• exUndoRedoAddElement(13) ~ <b>"AddElement;ELEMENTID"</b>, indicates that a new element has been created</li><li>• exUndoRedoRemoveElement(14) ~ <b>"RemoveElement;ELEMENTID"</b>, indicates that an element has been removed</li><li>• exUndoRedoMoveElement(15) ~ <b>"MoveElement;ELEMENTID"</b>, indicates that an element has been moved or resized</li><li>• exUndoRedoUpdateElement(16) ~ <b>"UpdateElement;ELEMENTID"</b>, indicates that one or more properties of the element has been updated, using the <a href="#">StartUpdateElement</a> / <a href="#">EndUpdateElement</a> methods</li><li>• exUndoRedoAddLink(10) ~ <b>"AddLink;LINKID"</b>, indicates that a new link has been created</li><li>• exUndoRedoRemoveLink(11) ~ <b>"RemoveLink;LINKID"</b>, indicates that a link has been removed</li><li>• exUndoRedoUpdateLink(12) ~ <b>"UpdateLink;LINKID"</b>, specifies that one of more properties of the link has been updated, using the <a href="#">StartUpdateLink</a> / <a href="#">EndUpdateLink</a> methods</li></ul> <p>For instance, UndoListAction(12) shows only AddElement actions in the undo stack.</p>
	<p>[optional] A long expression that indicates the number of actions being listed. If missing or -1, all actions are listed. For instance, UndoListAction(12,1) shows only the last AddElement action being added to the undo stack</p>



String	A String expression that lists the Undo actions that may be performed.
--------	------------------------------------------------------------------------

The `UndoListAction` property lists the Undo actions that can be performed in the control. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. Use the [UndoRemoveAction](#) method to remove the last actions from the undo queue. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked. The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed.

The records of the Undo/Redo queue may contain actions in the following format:

- **"AddElement;ELEMENTID"**, indicates that a new element has been created
- **"RemoveElement;ELEMENTID"**, indicates that an element has been removed
- **"MoveElement;ELEMENTID"**, indicates that an element has been moved or resized
- **"UpdateElement;ELEMENTID"**, indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- **"AddLink;LINKID"**, indicates that a new link has been created
- **"RemoveLink;LINKID"**, indicates that a link has been removed
- **"UpdateLink;LINKID"**, specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- **"StartBlock"**, specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- **"EndBlock"**, specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

Here's a sample how the result of `UndoListAction` method looks like:

```
StartBlock
MoveElement;3
AddElement;3
EndBlock
MoveElement;B
UpdateLink;Akak
AddLink;1
UpdateElement;B
AddElement;2
UpdateElement;A
```



# property Surface.UndoRedoQueueLength as Long

Gets or sets the maximum number of Undo/Redo actions that may be stored to the surface's queue.

Type	Description
Long	A Long expression that specifies the length of the Undo/Redo queue. If -1, the queue is unlimited, 0 allows no entries in the Undo/Redo queue (Undo/Redo is disabled).

By default, the UndoRedoQueueLength property is -1. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. Use the UndoRedoQueueLength property to specify the number of entries that Undo/Redo queue may store. For instance, if the UndoRedoQueueLength property is 1, the control retains only the last chart operation. Changing the UndoRedoQueueLength property may change the current Undo/Redo queue based on the new length. The length being specified, does not affect the blocks in the queue. A block may hold multiple Undo/Redo actions. Use the [GroupUndoRedoActions](#) method to group two or more entries in the Undo/Redo queue in a single block, so when a next Undo/Redo operation is performed, multiple actions may occur. For instance, moving several elements in the same time ( multiple elements selection ) is already recorded as a single block.

# method Surface.UndoRemoveAction([Action as Variant], [Count as Variant])

Removes the last undo actions that can be performed on the surface.

Type	Description
Action as Variant	<p>[optional] A long expression that specifies the action being remove. If missing or -1, all actions are removed.</p> <p>The Action parameter can be one of the following:</p> <ul style="list-style-type: none"><li>• exUndoRedoAddElement(13) ~ <b>"AddElement;ELEMENTID"</b>, indicates that a new element has been created</li><li>• exUndoRedoRemoveElement(14) ~ <b>"RemoveElement;ELEMENTID"</b>, indicates that an element has been removed</li><li>• exUndoRedoMoveElement(15) ~ <b>"MoveElement;ELEMENTID"</b>, indicates that an element has been moved or resized</li><li>• exUndoRedoUpdateElement(16) ~ <b>"UpdateElement;ELEMENTID"</b>, indicates that one or more properties of the element has been updated, using the <a href="#">StartUpdateElement</a> / <a href="#">EndUpdateElement</a> methods</li><li>• exUndoRedoAddLink(10) ~ <b>"AddLink;LINKID"</b>, indicates that a new link has been created</li><li>• exUndoRedoRemoveLink(11) ~ <b>"RemoveLink;LINKID"</b>, indicates that a link has been removed</li><li>• exUndoRedoUpdateLink(12) ~ <b>"UpdateLink;LINKID"</b>, specifies that one of more properties of the link has been updated, using the <a href="#">StartUpdateLink</a> / <a href="#">EndUpdateLink</a> methods</li></ul> <p>For instance, UndoRemoveAction(12) removes only AddElement actions from the undo stack.</p>
	<p>[optional] A long expression that indicates the number of actions to remove. If missing or -1, all actions are removed. For instance, UndoRemoveAction(12,1) removes only the last AddElement action from the undo stack</p>

Use the `UndoRemoveAction` method to remove the last action from the undo queue. Use the `UndoRemoveAction()` ( with no parameters ) to remove all undo actions. The [RedoRemoveAction](#) method removes the first action to be performed if the Redo method is invoked. The [AllowUndoRedo](#) property enables or disables the Undo/Redo feature. The [UndoListAction](#) property lists the Undo actions that can be performed in the control. The [RedoListAction](#) property lists the Redo actions that can be performed in the control. The [LayoutStartChanging](#)(exUndo/exRedo) / [LayoutEndChanging](#)(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddElement**;ELEMENTID", indicates that a new element has been created
- "**RemoveElement**;ELEMENTID", indicates that an element has been removed
- "**MoveElement**;ELEMENTID", indicates that an element has been moved or resized
- "**UpdateElement**;ELEMENTID", indicates that one or more properties of the element has been updated, using the [StartUpdateElement](#) / [EndUpdateElement](#) methods
- "**AddLink**;LINKID", indicates that a new link has been created
- "**RemoveLink**;LINKID", indicates that a link has been removed
- "**UpdateLink**;LINKID", specifies that one of more properties of the link has been updated, using the [StartUpdateLink](#) / [EndUpdateLink](#) methods

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins (initiated by [StartBlockUndoRedo](#) method)
- "**EndBlock**", specifies that a block of operations ends (initiated by [EndBlockUndoRedo](#) method)

# method Surface.UnselectAll ()

Unselects all elements in the control.

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

Use the UnselectAll method to unselect all elements on the surface. The [SelectAll](#) method selects all elements in the chart. The [SelectionChanged](#) event occurs once a new element is selected or unselected. The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The SelectObjectStyle property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface.

# property Surface.Version as String

Retrieves the control's version.

Type	Description
String	A String expression that specifies the version of the control you are running.

The Version property specifies the version of the control you are running.

# property Surface.VisualAppearance as Appearance

Retrieves the control's appearance.

Type	Description
<a href="#">Appearance</a>	The Appearance object holds a collection of skins.

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. You can use the [VisualDesign](#) property to design the visual appearance of the control at design mode.





# property Surface.VisualDesign as String

Invokes the control's VisualAppearance designer.

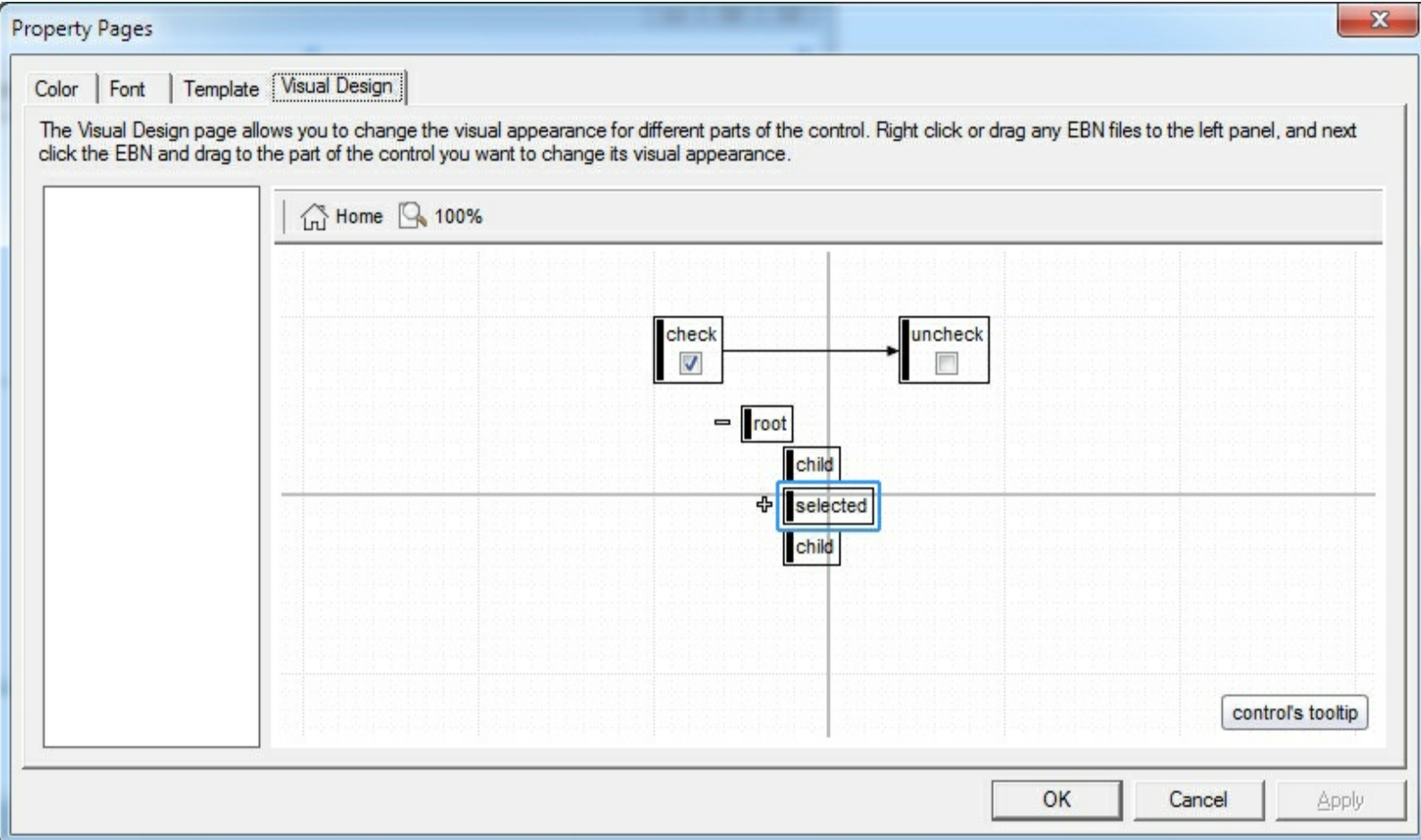
Type	Description
String	A String expression that encodes the control's Visual Appearance.

By default, the VisualDesign property is "". The VisualDesign property helps you to define fast and easy the control's visual appearance using the XP-Theme elements or [EBN](#) objects. The VisualDesign property can be accessed on design mode, and it can be used to design the visual appearance of different parts of the control by drag and drop XP or EBN elements. The VisualAppearance designer returns an encoded string that can be used to define different looks, just by calling the VisualDesign = encoded\_string. If you require removing the current visual appearance, you can call the VisualDesign on "" ( empty string ). The VisualDesign property encodes EBN or XP-Theme nodes, using the [Add](#) method of the [Appearance](#) collection being accessed through the [VisualAppearance](#) property.

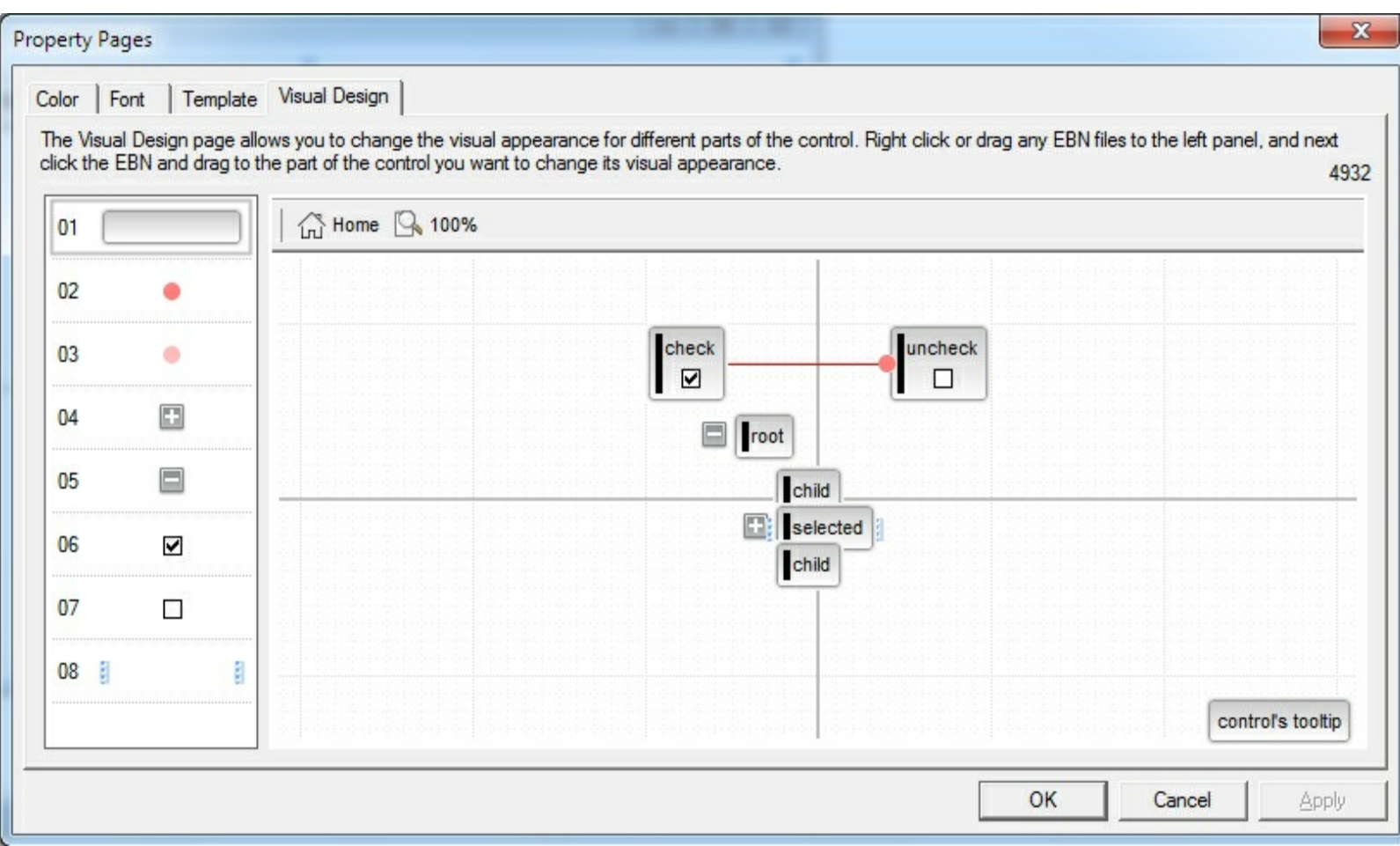
- For the /COM version, click the control in Design mode, select the Properties, and choose the "Visual Design" page.
- For the /NET version, select the VisualDesign property in the Properties browser, and then click ... so the "Visual Design" page is displayed.
- The /WPF version does not provide a VisualAppearance designer, instead you can use the values being generated by the /COM or /NET to apply the same visual appearance.
- Click here  to watch a movie on how you define the control's visual appearance using the XP-Theme
- Click here  to watch a movie on how you define the control's visual appearance using the EBN files.

The left panel, should be user to add your EBN or XP-Theme elements. Once you add them drag and drop the EBN or XP-Theme element from the left side to the part which visual appearance you want to change.

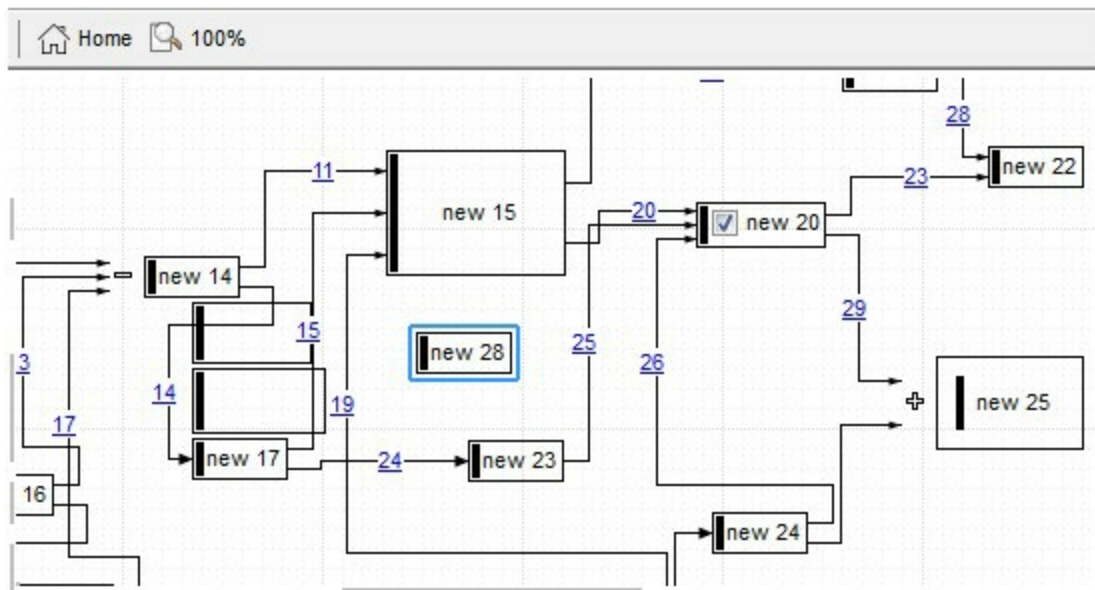
The following picture shows the control's VisualDesign form ( empty ):



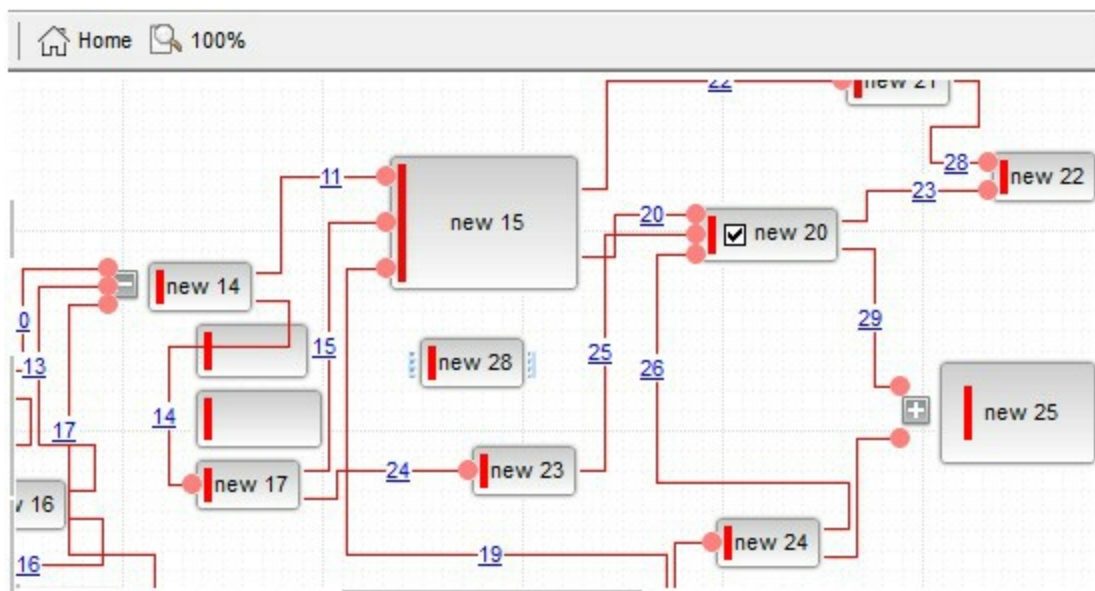
The following picture shows the control's VisualDesign form after applying some EBN objects:



If running the empty control we get the following picture:



If running the control using the code being generated by the VisualAppearance designer we get:



# property Surface.Zoom as Double

Specifies the current zooming factor of the surface.

Type	Description
Double	A Numeric expression that specifies the current zooming factor of the surface.

The Zoom property specifies the surface's current zooming factor. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface. The [ZoomLevels](#) property specifies the zooming factors to be displayed on the control's toolbar. The [ZoomMin](#), [ZoomMax](#) and [ZoomStep](#) determines the range of the zooming to be used, if the [ZoomLevels](#) property is empty. Use the [ToolBarFormat](#) property to customize the control's toolbar. The ZoomLevels property specifies the zooming factors to be displayed on the control's toolbar. The [AllowZoomSurface](#) property specifies the combination of keys that allows the user to magnify or shrink the surface. The [AllowZoomWheelSurface](#) property specifies whether the user can zoom the surface by rotating the mouse wheel.

The following samples shows how you can prevent zooming the surface:

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

```
With Surface1
    .AllowZoomSurface = 0
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB6

```
With Surface1
    .AllowZoomSurface = exDisallow
    .AllowZoomWheelSurface = False
    .ToolBarFormat = "-1,100"
End With
```

## VB.NET

```
With Exsurface1
    .AllowZoomSurface = exontrol.EXSURFACELib.AllowKeysEnum.exDisallow
    .AllowZoomWheelSurface = False
```

```
.ToolBarFormat = "-1,100"  
End With
```

## VB.NET for /COM

```
With AxSurface1  
    .AllowZoomSurface = EXSURFACELib.AllowKeysEnum.exDisallow  
    .AllowZoomWheelSurface = False  
    .ToolBarFormat = "-1,100"  
End With
```

## C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control  
    Library'  
  
    #import <ExSurface.dll>  
    using namespace EXSURFACELib;  
*/  
EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-  
> GetControlUnknown();  
spSurface1->PutAllowZoomSurface(EXSURFACELib::exDisallow);  
spSurface1->PutAllowZoomWheelSurface(VARIANT_FALSE);  
spSurface1->PutToolBarFormat(L"-1,100");
```

## C++ Builder

```
Surface1->AllowZoomSurface = Exsurfacelib_tlb::AllowKeysEnum::exDisallow;  
Surface1->AllowZoomWheelSurface = false;  
Surface1->ToolBarFormat = L"-1,100";
```

## C#

```
exsurface1.AllowZoomSurface = excontrol.EXSURFACELib.AllowKeysEnum.exDisallow;  
exsurface1.AllowZoomWheelSurface = false;
```

```
exsurface1.ToolBarFormat = "-1,100";
```

## JavaScript

```
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"  
id="Surface1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
    Surface1.AllowZoomSurface = 0;  
    Surface1.AllowZoomWheelSurface = false;  
    Surface1.ToolBarFormat = "-1,100";  
</SCRIPT>
```

## C# for /COM

```
axSurface1.AllowZoomSurface = EXSURFACELib.AllowKeysEnum.exDisallow;  
axSurface1.AllowZoomWheelSurface = false;  
axSurface1.ToolBarFormat = "-1,100";
```

## X++ (Dynamics Ax 2009)

```
public void init()  
{  
    ;  
  
    super();  
  
    exsurface1.AllowZoomSurface(0/*exDisallow*/);  
    exsurface1.AllowZoomWheelSurface(false);  
    exsurface1.ToolBarFormat("-1,100");  
}
```

## Delphi 8 (.NET only)

```
with AxSurface1 do  
begin  
    AllowZoomSurface := EXSURFACELib.AllowKeysEnum.exDisallow;
```

```
AllowZoomWheelSurface := False;  
ToolBarFormat := '-1,100';  
end
```

## Delphi (standard)

```
with Surface1 do  
begin  
    AllowZoomSurface := EXSURFACELib_TLB.exDisallow;  
    AllowZoomWheelSurface := False;  
    ToolBarFormat := '-1,100';  
end
```

## VFP

```
with thisform.Surface1  
    .AllowZoomSurface = 0  
    .AllowZoomWheelSurface = .F.  
    .ToolBarFormat = "-1,100"  
endwith
```

## dBASE Plus

```
local oSurface  
  
oSurface = form.Activex1.nativeObject  
oSurface.AllowZoomSurface = 0  
oSurface.AllowZoomWheelSurface = false  
oSurface.ToolBarFormat = "-1,100"
```

## XBasic (Alpha Five)

```
Dim oSurface as P  
  
oSurface = topparent:CONTROL_ACTIVEX1.activex  
oSurface.AllowZoomSurface = 0  
oSurface.AllowZoomWheelSurface = .f.  
oSurface.ToolBarFormat = "-1,100"
```

## Visual Objects

```
oDCOCX_Exontrol1:AllowZoomSurface := exDisallow  
oDCOCX_Exontrol1:AllowZoomWheelSurface := false  
oDCOCX_Exontrol1:ToolBarFormat := "-1,100"
```

## PowerBuilder

```
OleObject oSurface  
  
oSurface = ole_1.Object  
oSurface.AllowZoomSurface = 0  
oSurface.AllowZoomWheelSurface = false  
oSurface.ToolBarFormat = "-1,100"
```

## Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Set ComAllowZoomSurface to OLEexDisallow  
    Set ComAllowZoomWheelSurface to False  
    Set ComToolBarFormat to "-1,100"  
End_Procedure
```

## XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
    LOCAL oForm  
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
    LOCAL oSurface
```



```

oForm := XbpDialog():new( AppDesktop() )
oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oSurface := XbpActiveXControl():new( oForm:drawingArea )
oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
oSurface:create(,, {10,60},{610,370} )

oSurface:AllowZoomSurface := 0/*exDisallow*/
oSurface:AllowZoomWheelSurface := .F.
oSurface:ToolBarFormat := "-1,100"

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

# property Surface.ZoomLevels as String

Specifies the list of zooming factors to be displayed on the control's toolbar.

Type	Description
String	A String expression that specifies the zoom factors available on the surface, separated by comma characters.

By default, the ZoomLevels property is "25,35,50,75,100,150,200,300,400". The ZoomLevels property specifies the zooming factors to be displayed on the control's toolbar. If the ZoomLevels property is empty, the [ZoomMin](#), [ZoomMax](#) and [ZoomStep](#) determines the range of the zooming to be used. Use the [ToolBarFormat](#) property to customize the control's toolbar. The [Zoom](#) property specifies the surface's current zooming factor. The [AllowZoomSurface](#) property specifies the combination of keys that allows the user to magnify or shrink the surface. The [AllowZoomWheelSurface](#) property specifies whether the user can zoom the surface by rotating the mouse wheel. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface.

# property Surface.ZoomMax as Double

Specifies the maximum zooming factor of the surface.

Type	Description
Double	A Numeric expression that specifies the max value for the Zoom property.

By default, the ZoomMax property is 400. the [ZoomMin](#), ZoomMax and [ZoomStep](#) determines the range of the zooming to be used, if the [ZoomLevels](#) property is empty. Use the [ToolBarFormat](#) property to customize the control's toolbar. The ZoomLevels property specifies the zooming factors to be displayed on the control's toolbar. The [Zoom](#) property specifies the surface's current zooming factor. The [AllowZoomSurface](#) property specifies the combination of keys that allows the user to magnify or shrink the surface. The [AllowZoomWheelSurface](#) property specifies whether the user can zoom the surface by rotating the mouse wheel. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface.

# property Surface.ZoomMin as Double

Specifies the minimum zooming factor of the surface.

Type	Description
Double	A Numeric expression that specifies the min value for the Zoom property.

By default, the ZoomMin property is 20. the ZoomMin, [ZoomMax](#) and [ZoomStep](#) determines the range of the zooming to be used, if the [ZoomLevels](#) property is empty. Use the [ToolBarFormat](#) property to customize the control's toolbar. The ZoomLevels property specifies the zooming factors to be displayed on the control's toolbar. The [Zoom](#) property specifies the surface's current zooming factor. The [AllowZoomSurface](#) property specifies the combination of keys that allows the user to magnify or shrink the surface. The [AllowZoomWheelSurface](#) property specifies whether the user can zoom the surface by rotating the mouse wheel. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface.

# property Surface.ZoomStep as Double

Specifies the step to increase or decrease the zooming factor of the surface, while the user rotates the mouse wheel.

Type	Description
Double	A Numeric expression that specifies the step value for the Zoom property.

By default, the ZoomStep property is 10. the [ZoomMin](#), [ZoomMax](#) and ZoomStep determines the range of the zooming to be used, if the [ZoomLevels](#) property is empty. Use the [ToolBarFormat](#) property to customize the control's toolbar. The ZoomLevels property specifies the zooming factors to be displayed on the control's toolbar. The [Zoom](#) property specifies the surface's current zooming factor. The [AllowZoomSurface](#) property specifies the combination of keys that allows the user to magnify or shrink the surface. The [AllowZoomWheelSurface](#) property specifies whether the user can zoom the surface by rotating the mouse wheel. The control fires the [LayoutStartChanging](#)(exSurfaceZoom) / [LayoutEndChanging](#)(exSurfaceZoom) event when the user zooms the surface.

# ExSurface events

**Tip** The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {AC1DF7F4-0919-4364-8167-2F9B5155EA4B}. The object's program identifier is: "Exontrol.Surface". The /COM object module is: "ExSurface.dll"

The exGrid component supports the following events:

Name	Description
<a href="#">AddElement</a>	A new element has been added to the surface.
<a href="#">AddLink</a>	A new link has been added to the links collection.
<a href="#">AllowLink</a>	Occurs when the user links an element with another element.
<a href="#">AnchorClick</a>	Occurs when an anchor element is clicked.
<a href="#">CheckElement</a>	The element's Checked property has been changed.
<a href="#">Click</a>	Occurs when the user presses and then releases the left mouse button over the control.
<a href="#">CreateElement</a>	The user creates at runtime a new element.
<a href="#">CreateLink</a>	The user creates at runtime a new link.
<a href="#">DbClick</a>	Occurs when the user dblclk the left mouse button over an object.
<a href="#">Event</a>	Notifies the application once the control fires an event.
<a href="#">ExpandElement</a>	The element is expanded or collapsed.
<a href="#">HandCursorClick</a>	The uses clicks a part of the element that shows the had cursor.
<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="#">LayoutEndChanging</a>	Notifies your application once the control's layout has been changed.
<a href="#">LayoutStartChanging</a>	Occurs when the control's layout is about to be changed.
<a href="#">MouseDown</a>	Occurs when the user presses a mouse button.
<a href="#">MouseMove</a>	Occurs when the user moves the mouse.
<a href="#">MouseUp</a>	Occurs when the user releases a mouse button.

[OLECompleteDrag](#)

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled

[OLEDragDrop](#)

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

[OLEDragOver](#)

Occurs when one component is dragged over another.

[OleEvent](#)

Occurs once an inside control fires an event.

[OLEGiveFeedback](#)

Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.

[OLESetData](#)

Occurs on a drag source when a drop target calls the GetData method and there is no data in a specified format in the OLE drag-and-drop DataObject.

[OLEStartDrag](#)

Occurs when the OLEDrag method is called.

[ParentChangeElement](#)

The element's parent is changed.

[RClick](#)

Occurs once the user right clicks the control.

[RemoveElement](#)

An element has been removed from the surface.

[RemoveLink](#)

The link is removed from the links collection.

[SelectionChanged](#)

Notifies your application that the control's selection has been changed.

[ToolBarAnchorClick](#)

Occurs when an anchor element is clicked, on the control's toolbar.

[ToolBarClick](#)

Occurs when the user clicks a button in the toolbar.

## event AddElement (Element as Element)

A new element has been added to the surface.

Type	Description
Element as <a href="#">Element</a>	An Element object being added to the Elements collection.

The AddElement event notifies your application once a new element has been added to the [Elements](#) collection. The [Add](#) method adds a new element to the Elements collection. The [CreateElement](#) event notifies your application when the user creates at runtime the element on the surface. You can use the AddElement event to change properties of the added element to default values or to associate any extra data to the added element.

The order of the events when the user creates the element at runtime is:

- [LayoutStartChanging](#)(exCreateObject), the user clicks on the surface
- AddElement, adds the new element to the Elements collection
- [CreateElement](#), the user ends creating the object
- [LayoutEndChanging](#)(exCreateObject), the user un-clicks the surface

The AddElement event may be called during the [LoadXML](#) method.

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

```
C# private void AddElement(object sender,exontrol.EXSURFACELib.Element Element)
{
}
```

```
VB Private Sub AddElement(ByVal sender As System.Object,ByVal Element As
exontrol.EXSURFACELib.Element) Handles AddElement
End Sub
```

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

```
C# private void AddElement(object sender,
AxEXSURFACELib._ISurfaceEvents_AddElementEvent e)
{
}
```

```
C++ void OnAddElement(LPDISPATCH Element)
{
}
```



C++  
Builder

```
void __fastcall AddElement(TObject *Sender,Exsurfacelib_tlb::IElement *Element)
{
}
```

Delphi

```
procedure AddElement(ASender: TObject; Element : IElement);
begin
end;
```

Delphi 8  
(.NET  
only)

```
procedure AddElement(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_AddElementEvent);
begin
end;
```

Powe...

```
begin event AddElement(oleobject Element)
end event AddElement
```

VB.NET

```
Private Sub AddElement(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_AddElementEvent) Handles AddElement
End Sub
```

VB6

```
Private Sub AddElement(ByVal Element As EXSURFACELibCtl.IElement)
End Sub
```

VBA

```
Private Sub AddElement(ByVal Element As Object)
End Sub
```

VFP

```
LPARAMETERS Element
```

Xbas...

```
PROCEDURE OnAddElement(oSurface,Element)
RETURN
```

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

Java...

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

VBSc...

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

```
Function AddElement(Element)
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComAddElement Variant IElement
    Forward Send OnComAddElement IElement
End_Procedure
```

Visual  
Objects

```
METHOD OCX_AddElement(Element) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddElement(COM _Element)
{
}
```

XBasic

```
function AddElement as v (Element as OLE::Exontrol.Surface.1::IElement)
end function
```

dBASE

```
function nativeObject_AddElement(Element)
return
```

# event AddLink (Link as Link)

A new link has been added to the links collection.

Type	Description
Link as <a href="#">Link</a>	A Link object being added to the Links collection.

The AddLink event notifies your application once a new link has been added to the [Links](#) collection. The [Add](#) method adds a new link to the Links collection. The [CreateLink](#) event notifies your application when the user links two elements on the surface. You can use the AddLink event to change properties of the added link to default values or to associate any extra data to the added link. The [AllowLink](#) event occurs when user links two elements to specify whether the link is allowed.

The order of the events when the user links two elements at runtime is:

- [LayoutStartChanging](#)(exLinkObjects), the user clicks on the surface
- [AllowLink](#), occurs to specify whether the link between two elements is possible.
- AddLink, adds the new link to the Links collection
- [CreateLink](#), the user ends creating the link
- [LayoutEndChanging](#)(exLinkObjects), the user un-clicks the surface

The AddLink event may be called during the [LoadXML](#) method.

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

```
C# private void AddLink(object sender,exontrol.EXSURFACELib.Link Link)
{
}
```

```
VB Private Sub AddLink(ByVal sender As System.Object,ByVal Link As
exontrol.EXSURFACELib.Link) Handles AddLink
End Sub
```

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

```
C# private void AddLink(object sender,
AxEXSURFACELib._ISurfaceEvents_AddLinkEvent e)
{
}
```

```
C++ void OnAddLink(LPDISPATCH Link)
```

```
{  
}
```

**C++ Builder**

```
void __fastcall AddLink(TObject *Sender, Exsurfacelib_tlb::ILink *Link)  
{  
}
```

**Delphi**

```
procedure AddLink(ASender: TObject; Link : ILink);  
begin  
end;
```

**Delphi 8  
(.NET only)**

```
procedure AddLink(sender: System.Object; e:  
AxEXSURFACELib._ISurfaceEvents_AddLinkEvent);  
begin  
end;
```

**Powe...**

```
begin event AddLink(oleobject Link)  
end event AddLink
```

**VB.NET**

```
Private Sub AddLink(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_AddLinkEvent) Handles AddLink  
End Sub
```

**VB6**

```
Private Sub AddLink(ByVal Link As EXSURFACELibCtl.ILink)  
End Sub
```

**VBA**

```
Private Sub AddLink(ByVal Link As Object)  
End Sub
```

**VFP**

```
LPARAMETERS Link
```

**Xbas...**

```
PROCEDURE OnAddLink(oSurface, Link)  
RETURN
```

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

**Java...**

```
<SCRIPT EVENT="AddLink(Link)" LANGUAGE="JScript">
```

```
</SCRIPT>
```

VBSc...

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

Visual  
Data...

```
Procedure OnComAddLink Variant ILink  
    Forward Send OnComAddLink ILink  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_AddLink(Link) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_AddLink(COM _Link)  
{  
}
```

XBasic

```
function AddLink as v (Link as OLE::Exontrol.Surface.1::ILink)  
end function
```

dBASE

```
function nativeObject_AddLink(Link)  
return
```

# event AllowLink (ElementFrom as Element, ElementTo as Element, ByRef Cancel as Boolean)

Occurs when the user links an element with another element.

Type	Description
ElementFrom as <a href="#">Element</a>	An Element object that specifies where the link starts from.
ElementTo as <a href="#">Element</a>	An Element object that specifies where the link ends to.
Cancel as Boolean	(By Reference) A Boolean expression that specifies whether the operation is canceled or allowed, or if the link will or will not be added.

The AllowLink event occurs when user links two elements to specify whether the link is allowed. For instance, you can use the [PathTo](#) property of the Element object to check if there is a path from an element to another, so avoid cycles. The [CreateLink](#) event occurs when the user links two elements on the surface. The [AllowLinkObjects](#) property specifies the keys combination to let the user links two elements on the surface, the [ElementFrom](#) property specifies the element where the link starts, where the [ElementTo](#) property specifies ending element of the link. Prior to CreateLink event the control fires the AddLink event that indicates that the link has been added to the [Links](#) collection. You can use the [Remove](#) method to remove the link. The [Background](#)(exLinkObjectsInvalidColor) property specifies the color to show the invalid link. The [Background](#)(exLinkObjectsValidColor) property specifies the color to show the valid link.

The following VB sample prevents adding cycles to the chart:

```
Private Sub Surface1_AllowLink(ByVal ElementFrom As EXSURFACELibCtl.IElement, ByVal
ElementTo As EXSURFACELibCtl.IElement, Cancel As Boolean)
    Cancel = ElementTo.PathTo(ElementFrom)
End Sub
```

The order of the events when the user links two elements at runtime is:

- [LayoutStartChanging](#)(exLinkObjects), the user clicks on the surface
- AllowLink, occurs to specify whether the link between two elements is possible.
- [AddLink](#), adds the new link to the Links collection
- [CreateLink](#), the user ends creating the link
- [LayoutEndChanging](#)(exLinkObjects), the user un-clicks the surface

The AllowLink event is not called during the [LoadXML](#) method.

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

```
C# private void AllowLink(object sender,exontrol.EXSURFACELib.Element
    ElementFrom,exontrol.EXSURFACELib.Element ElementTo,ref bool Cancel)
{
}
```

```
VB Private Sub AllowLink(ByVal sender As System.Object,ByVal ElementFrom As
    exontrol.EXSURFACELib.Element,ByVal ElementTo As
    exontrol.EXSURFACELib.Element,ByRef Cancel As Boolean) Handles AllowLink
End Sub
```

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

```
C# private void AllowLink(object sender,
    AxEXSURFACELib._ISurfaceEvents_AllowLinkEvent e)
{
}
```

```
C++ void OnAllowLink(LPDISPATCH ElementFrom,LPDISPATCH ElementTo,BOOL FAR*
    Cancel)
{
}
```

```
C++ Builder void __fastcall AllowLink(TObject *Sender,Exsurfacelib_tlb::IElement
    *ElementFrom,Exsurfacelib_tlb::IElement *ElementTo,VARIANT_BOOL * Cancel)
{
}
```

```
Delphi procedure AllowLink(ASender: TObject; ElementFrom : IElement;ElementTo :
    IElement;var Cancel : WordBool);
begin
end;
```

```
Delphi 8 (.NET only) procedure AllowLink(sender: System.Object; e:
    AxEXSURFACELib._ISurfaceEvents_AllowLinkEvent);
begin
end;
```

**Powe...** begin event AllowLink(oleobject ElementFrom,oleobject ElementTo,boolean Cancel)  
end event AllowLink

**VB.NET** Private Sub AllowLink(ByVal sender As System.Object, ByVal e As AxEXSURFACELib.\_ISurfaceEvents\_AllowLinkEvent) Handles AllowLink  
End Sub

**VB6** Private Sub AllowLink(ByVal ElementFrom As EXSURFACELibCtl.IElement,ByVal ElementTo As EXSURFACELibCtl.IElement,Cancel As Boolean)  
End Sub

**VBA** Private Sub AllowLink(ByVal ElementFrom As Object,ByVal ElementTo As Object,Cancel As Boolean)  
End Sub

**VFP** LPARAMETERS ElementFrom,ElementTo,Cancel

**Xbas...** PROCEDURE OnAllowLink(oSurface,ElementFrom,ElementTo,Cancel)  
RETURN

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

**Java...** <SCRIPT EVENT="AllowLink(ElementFrom,ElementTo,Cancel)"  
LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function AllowLink(ElementFrom,ElementTo,Cancel)  
End Function  
</SCRIPT>

**Visual Data...** Procedure OnComAllowLink Variant IElementFrom Variant IElementTo Boolean IICancel  
Forward Send OnComAllowLink IElementFrom IElementTo IICancel  
End\_Procedure



Visual  
Objects

METHOD OCX\_AllowLink(ElementFrom,ElementTo,Cancel) CLASS MainDialog  
RETURN NIL

X++

```
void onEvent_AllowLink(COM _ElementFrom,COM _ElementTo,COMVariant  
/*bool*/ _Cancel)  
{  
}
```

XBasic

```
function AllowLink as v (ElementFrom as  
OLE::Exontrol.Surface.1::IElement,ElementTo as  
OLE::Exontrol.Surface.1::IElement,Cancel as L)  
end function
```

dBASE

```
function nativeObject_AllowLink(ElementFrom,ElementTo,Cancel)  
return
```

# 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 [Caption](#) or [ExtraCaption](#) property to display hyperlinks or anchors in the element. Use the [Caption](#) property to assign a caption/hyperlink to a link.

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,
AxEXSURFACELib._ISurfaceEvents_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: AxEXSURFACELib.\_ISurfaceEvents\_AnchorClickEvent);  
begin  
end;

**PowerBuilder** begin event AnchorClick(string AnchorID,string Options)  
end event AnchorClick

**VB.NET** Private Sub AnchorClick(ByVal sender As System.Object, ByVal e As AxEXSURFACELib.\_ISurfaceEvents\_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

**Xbase++** PROCEDURE OnAnchorClick(oSurface,AnchorID,Options)  
RETURN

Syntax for AnchorClick event, **/COM** 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 IIAnchorID String IIOptions  
Forward Send OnComAnchorClick IIAnchorID IIOptions  
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 CheckElement (Element as Element)

The element's Checked property has been changed.

Type	Description
Element as <a href="#">Element</a>	An Element object that specifies the element being checked or un-checked.

The CheckElement event notifies your application once the user clicks the element's check-box. The [ShowCheckBox](#) property specifies whether the element displays the check-box. The [Checked](#) property indicates whether the element is checked or unchecked. Use the [CheckBoxAlign](#) property to align the element's checkbox. The [Background](#)(exCheckBoxState0) Specifies the visual appearance for the check box in 0 state. The [Background](#)(exCheckBoxState1) Specifies the visual appearance for the check box in 1 state.

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

C#private void CheckElement(object sender,exontrol.EXSURFACELib.Element Element)  
{  
}

VBPrivate Sub CheckElement(ByVal sender As System.Object,ByVal Element As exontrol.EXSURFACELib.Element) Handles CheckElement  
End Sub

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

C#private void CheckElement(object sender,  
AxEXSURFACELib.\_ISurfaceEvents\_CheckElementEvent e)  
{  
}

C++void OnCheckElement(LPDISPATCH Element)  
{  
}

C++ Buildervoid \_\_fastcall CheckElement(TObject \*Sender,Exsurfacelib\_tlb::IElement \*Element)  
{  
}

**Delphi** procedure CheckElement(ASender: TObject; Element : IElement);  
begin  
end;

**Delphi 8  
(.NET  
only)** procedure CheckElement(sender: System.Object; e:  
AxEXSURFACELib.\_ISurfaceEvents\_CheckElementEvent);  
begin  
end;

**Powe...** begin event CheckElement(oleobject Element)  
end event CheckElement

**VB.NET** Private Sub CheckElement(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_CheckElementEvent) Handles CheckElement  
End Sub

**VB6** Private Sub CheckElement(ByVal Element As EXSURFACELibCtl.IElement)  
End Sub

**VBA** Private Sub CheckElement(ByVal Element As Object)  
End Sub

**VFP** LPARAMETERS Element

**Xbas...** PROCEDURE OnCheckElement(oSurface,Element)  
RETURN

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

**Java...** <SCRIPT EVENT="CheckElement(Element)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function CheckElement(Element)  
End Function  
</SCRIPT>

Visual  
Data...

```
Procedure OnComCheckElement Variant IElement  
    Forward Send OnComCheckElement IElement  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_CheckElement(Element) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_CheckElement(COM _Element)  
{  
}
```

XBasic

```
function CheckElement as v (Element as OLE::Exontrol.Surface.1::IElement)  
end function
```

dBASE

```
function nativeObject_CheckElement(Element)  
return
```

# event Click ()

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

## Type

## Description

The Click event notifies your application once the user clicks the surface. Use a [MouseDown](#) or [MouseUp](#) event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#) property returns the element and the hit-test code from the cursor. You can use the [Edit](#) method to edit the element's caption or extra caption. The [HandCursorClick](#) event notifies once the user clicks a part of the element ( which shows a hand cursor when the pointer hovers it ).

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

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

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

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

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

```
C++ void OnClick()
{
}
```

```
C++ Builder void __fastcall Click(TObject *Sender)
{
}
```

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



```
end;
```

Delphi 8  
(.NET  
only)

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

Power...

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

VB.NET

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

VB6

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

VBA

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

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnClick(oSurface)  
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 CreateElement (Element as Element)

The user creates at runtime a new element.

Type	Description
Element as <a href="#">Element</a>	An Element object being created.

The CreateElement event occurs when the user creates the element on the surface. The [AllowCreateObject](#) property specifies the keys combination to let the user creates the elements at runtime. Prior to CreateElement event the AddElement event is fired to notify that the element has been added to the Elements collection. The CreateElement event is not fired when you add programmatically the element calling the [Add](#) method. For instance, you can call the [Element.Edit](#) method during the CreateElement to let the user edits the element's caption once a new element is created. You can call the [Remove](#) method to remove the newly created element.

The order of the events when the user creates the element at runtime is:

- [LayoutStartChanging](#)(exCreateObject), the user clicks on the surface
- [AddElement](#), adds the new element to the Elements collection
- CreateElement, the user ends creating the object
- [LayoutEndChanging](#)(exCreateObject), the user un-clicks the surface

The CreateElement event is not called during the [LoadXML](#) method.

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

```
C# private void CreateElement(object sender,excontrol.EXSURFACELib.Element
    Element)
    {
    }
```

```
VB Private Sub CreateElement(ByVal sender As System.Object,ByVal Element As
    excontrol.EXSURFACELib.Element) Handles CreateElement
End Sub
```

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

```
C# private void CreateElement(object sender,
    AxEXSURFACELib._ISurfaceEvents_CreateElementEvent e)
    {
    }
```

**C++**

```
void OnCreateElement(LPDISPATCH Element)
{
}
```

**C++  
Builder**

```
void __fastcall CreateElement(TObject *Sender,Exsurfacelib_tlb::IElement *Element)
{
}
```

**Delphi**

```
procedure CreateElement(ASender: TObject; Element : IElement);
begin
end;
```

**Delphi 8  
(.NET  
only)**

```
procedure CreateElement(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_CreateElementEvent);
begin
end;
```

**Powe...**

```
begin event CreateElement(oleobject Element)
end event CreateElement
```

**VB.NET**

```
Private Sub CreateElement(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_CreateElementEvent) Handles CreateElement
End Sub
```

**VB6**

```
Private Sub CreateElement(ByVal Element As EXSURFACELibCtl.IElement)
End Sub
```

**VBA**

```
Private Sub CreateElement(ByVal Element As Object)
End Sub
```

**VFP**

```
LPARAMETERS Element
```

**Xbas...**

```
PROCEDURE OnCreateElement(oSurface,Element)
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComCreateElement Variant IElement
    Forward Send OnComCreateElement IElement
End_Procedure
```

Visual  
Objects

```
METHOD OCX_CreateElement(Element) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CreateElement(COM _Element)
{
}
```

XBasic

```
function CreateElement as v (Element as OLE::Exontrol.Surface.1::IElement)
end function
```

dBASE

```
function nativeObject_CreateElement(Element)
return
```

The following VB sample calls the [Edit](#) method once a new element is created:

```
Private Sub Surface1_CreateElement(ByVal Element As EXSURFACELibCtl.IElement)
    With Element
        .AutoSize = True
        .Caption = "new " & Surface1.Elements.Count
        .Edit exEditCaption
    End With
End Sub
```

The following VB sample creates an element that hosts the [Exontrol.Button](#) control:

```
Private Sub Surface1_CreateElement(ByVal Element As EXSURFACELibCtl.IElement)
    With Element
        .Type = exElementHostControl
        .ElementFormat = ""client""
        .Control = "Exontrol.Button"
        With .Object
            .Caption = "<sha ;;0>Button " & Surface1.Elements.Count
        End With
    End With
End Sub
```

# event CreateLink (Link as Link)

The user creates at runtime a new link.

Type	Description
Link as <a href="#">Link</a>	A Link object being created.

The CreateLink event occurs when the user links two elements on the surface. The [AllowLinkObjects](#) property specifies the keys combination to let the user links two elements on the surface, the [ElementFrom](#) property specifies the element where the link starts, where the [ElementTo](#) property specifies ending element of the link. Prior to CreateLink event the control fires the [AddLink](#) event that indicates that the link has been added to the [Links](#) collection. You can use the [Remove](#) method to remove the link. The [AllowLink](#) event occurs when user links two elements to specify whether the link is allowed. The [Background](#)( exLinkObjectsInvalidColor) property specifies the color to show the invalid link. The [Background](#)(exLinkObjectsValidColor) property specifies the color to show the valid link.

The order of the events when the user links two elements at runtime is:

- [LayoutStartChanging](#)(exLinkObjects), the user clicks on the surface
- [AllowLink](#), occurs to specify whether the link between two elements is possible.
- [AddLink](#), adds the new link to the Links collection
- CreateLink, the user ends creating the link
- [LayoutEndChanging](#)(exLinkObjects), the user un-clicks the surface

The CreateLink event is not called during the [LoadXML](#) method.

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

```
C# private void CreateLink(object sender,exontrol.EXSURFACELib.Link Link)
{
}
```

```
VB Private Sub CreateLink(ByVal sender As System.Object,ByVal Link As
exontrol.EXSURFACELib.Link) Handles CreateLink
End Sub
```

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

```
C# private void CreateLink(object sender,
AxEXSURFACELib._ISurfaceEvents_CreateLinkEvent e)
{
```

```
}
```

C++

```
void OnCreateLink(LPDISPATCH Link)
{
}
```

C++  
Builder

```
void __fastcall CreateLink(TObject *Sender,Exsurfacelib_tlb::ILink *Link)
{
}
```

Delphi

```
procedure CreateLink(ASender: TObject; Link : ILink);
begin
end;
```

Delphi 8  
(.NET  
only)

```
procedure CreateLink(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_CreateLinkEvent);
begin
end;
```

Powe...

```
begin event CreateLink(oleobject Link)
end event CreateLink
```

VB.NET

```
Private Sub CreateLink(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_CreateLinkEvent) Handles CreateLink
End Sub
```

VB6

```
Private Sub CreateLink(ByVal Link As EXSURFACELibCtl.ILink)
End Sub
```

VBA

```
Private Sub CreateLink(ByVal Link As Object)
End Sub
```

VFP

```
LPARAMETERS Link
```

Xbas...

```
PROCEDURE OnCreateLink(oSurface,Link)
RETURN
```



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

Java... <SCRIPT EVENT="CreateLink(Link)" LANGUAGE="JScript">  
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">  
Function CreateLink(Link)  
End Function  
</SCRIPT>

Visual  
Data... Procedure OnComCreateLink Variant ILink  
Forward Send OnComCreateLink ILink  
End\_Procedure

Visual  
Objects METHOD OCX\_CreateLink(Link) CLASS MainDialog  
RETURN NIL

X++ void onEvent\_CreateLink(COM \_Link)  
{  
}

XBasic function CreateLink as v (Link as OLE::Exontrol.Surface.1::ILink)  
end function

dBASE function nativeObject\_CreateLink(Link)  
return

## event DblClick (Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS)

Occurs when the user dblclk the left mouse button over an object.

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

The DblClick event is fired when the user double clicks the control. Use a [MouseDown](#) or [MouseUp](#) event procedure to specify actions that will occur when a mouse button is pressed or released. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#) property returns the element and the hit-test code from the cursor. You can use the [Edit](#) method to edit the element's caption or extra caption. Use the [AllowCreateObject](#) property to specify the keys combination so the user creates the elements in the surface. By default, the control creates a new element once the user double clicks the surface.

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

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

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

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

```
C# private void DblClick(object sender,
AxEXSURFACELib._ISurfaceEvents_DblClickEvent e)
{
}
```

**C++**

```
void OnDbClick(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:
AxEXSURFACELib._ISurfaceEvents_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
AxEXSURFACELib._ISurfaceEvents_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(oSurface,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.Surface.1::OLE_XPOS_PIXELS,Y
as OLE::Exontrol.Surface.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_DbClick(Shift,X,Y)
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. Each internal event of the control has an unique identifier. Use the <a href="#">EventParam(-2)</a> to display entire information about fired event ( such as name, identifier, and properties ). The EventParam(-1) retrieves the number of parameters of fired event

The Event notification occurs ANY time the control fires an event. *This is useful for X++, which does not support event with parameters passed by reference. Also, this could be useful for C++ Builder or Delphi, which does not handle properly the events with parameters of VARIANT type.*

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.

If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    print exsurface1.EventParam(-2).toString();
}
```

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

```
"MouseMove/-606( 1 , 0 , 145 , 36 )" VT_BSTR
"BeforeDrawPart/54( 2 , -1962866148 , =0 , =0 , =0 , =0 , =false )" VT_BSTR
"AfterDrawPart/55( 2 , -1962866148 , 0 , 0 , 0 , 0 )" VT_BSTR
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

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

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

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

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

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

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

```
C# private void Event(object sender, AxEXSURFACELib._ISurfaceEvents_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:
AxEXSURFACELib._ISurfaceEvents_EventEvent);
begin
end;
```

Power... begin event Event(long EventID)  
end event Event

VB.NET Private Sub Event(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_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(oSurface,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 lEventID  
Forward Send OnComEvent lEventID  
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 ExpandElement (Element as Element)

The element is expanded or collapsed.

Type	Description
Element as <a href="#">Element</a>	An Element being expanded or collapsed.

The ExpandElement event is fired when an element is expanded or collapsed. The element displays the expanding/collapsing glyph if the element contains child elements or there are links that starts from the element ( outgoing links, while the [ExpandLinkedElements](#) property is True ). The [Expanded](#) property specifies whether the element is expanded or collapsed. The [Children](#) property returns the list of child element of giving element. The [OutgoingLinks](#) property indicates the list of links that starts from specified element.

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

C#private void ExpandElement(object sender,exontrol.EXSURFACELib.Element Element)  
{  
}

VBPrivate Sub ExpandElement(ByVal sender As System.Object,ByVal Element As exontrol.EXSURFACELib.Element) Handles ExpandElement  
End Sub

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

C#private void ExpandElement(object sender,  
AxEXSURFACELib.\_ISurfaceEvents\_ExpandElementEvent e)  
{  
}

C++void OnExpandElement(LPDISPATCH Element)  
{  
}

C++ Buildervoid \_\_fastcall ExpandElement(TObject \*Sender,Exsurfacelib\_tlb::IElement \*Element)  
{  
}

**Delphi** procedure ExpandElement(ASender: TObject; Element : IElement);  
begin  
end;

**Delphi 8  
(.NET  
only)** procedure ExpandElement(sender: System.Object; e:  
AxEXSURFACELib.\_ISurfaceEvents\_ExpandElementEvent);  
begin  
end;

**Powe...** begin event ExpandElement(oleobject Element)  
end event ExpandElement

**VB.NET** Private Sub ExpandElement(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_ExpandElementEvent) Handles ExpandElement  
End Sub

**VB6** Private Sub ExpandElement(ByVal Element As EXSURFACELibCtl.IElement)  
End Sub

**VBA** Private Sub ExpandElement(ByVal Element As Object)  
End Sub

**VFP** LPARAMETERS Element

**Xbas...** PROCEDURE OnExpandElement(oSurface,Element)  
RETURN

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

**Java...** <SCRIPT EVENT="ExpandElement(Element)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function ExpandElement(Element)  
End Function  
</SCRIPT>

Visual  
Data...

```
Procedure OnComExpandElement Variant IElement  
    Forward Send OnComExpandElement IElement  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_ExpandElement(Element) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_ExpandElement(COM _Element)  
{  
}
```

XBasic

```
function ExpandElement as v (Element as OLE::Exontrol.Surface.1::IElement)  
end function
```

dBASE

```
function nativeObject_ExpandElement(Element)  
return
```

# event HandCursorClick (Element as Element, Hit as ShowHandCursorOnEnum, Key as Variant)

The uses clicks a part of the element that shows the had cursor.

Type	Description
Element as <a href="#">Element</a>	An Element object being clicked.
Hit as <a href="#">ShowHandCursorOnEnum</a>	A ShowHandCursorOnEnum expression that specifies the part of the element being clicked.
Key as Variant	A VARIANT expression that specifies the key associated with the part being clicked. For instance, if the Hit indicates the exShowHandCursorCheck flag, the Key parameter specifies the element's checkbox state.

The HandCursorClick event notifies once the user clicks a part of the element ( which shows a hand cursor when the pointer hovers it ). The [ShowHandCursorOn](#) property specifies the parts of the element that shows the hand cursor when the mouse-pointer hovers the part. The Hit parameter specifies the part of the element being clicked, while the Key parameter specifies a value associated with the part being clicked as listed below:

- exShowHandCursorCheck -> key specifies the [Element.Checked](#) property.
- exShowHandCursorAnchor -> key specifies the identifier of the anchor element such as <a id;options> anchor </a>
- exShowHandCursorPicture -> key specifies the name of the picture being clicked ( [HTMLPicture](#) property )
- exShowHandCursorIcon - key specifies the index of the icon being clicked ( [Images](#) method )

The above flags can be combined with the following flags:

- exShowHandCursorCaption, indicates that the part being clicked belong to the element's caption.
- exShowHandCursorExtraCaption, indicates that the part being clicked belong to the element's extra caption.
- exShowHandCursorPictures, indicates that the part being clicked belong to the element's pictures.
- exShowHandCursorExtraPictures, indicates that the part being clicked belong to the element's extra pictures.

The HandCursorClick event occurs also if an anchor element is clicked so you can handle the [AnchorClick](#) event too. You can use the [HitTestFromPoint](#) property to determine whether the cursor hovers the expand/collapse glyphs, the element's checkbox, picture and so on.

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

```
C# private void HandCursorClick(object sender,exontrol.EXSURFACELib.Element
Element,exontrol.EXSURFACELib.ShowHandCursorOnEnum Hit,object Key)
{
}
```

```
VB Private Sub HandCursorClick(ByVal sender As System.Object,ByVal Element As
exontrol.EXSURFACELib.Element,ByVal Hit As
exontrol.EXSURFACELib.ShowHandCursorOnEnum,ByVal Key As Object) Handles
HandCursorClick
End Sub
```

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

```
C# private void HandCursorClick(object sender,
AxEXSURFACELib._ISurfaceEvents_HandCursorClickEvent e)
{
}
```

```
C++ void OnHandCursorClick(LPDISPATCH Element,long Hit,VARIANT Key)
{
}
```

```
C++ Builder void __fastcall HandCursorClick(TObject *Sender,Exsurfacelib_tlb::IElement
*Element,Exsurfacelib_tlb::ShowHandCursorOnEnum Hit,Variant Key)
{
}
```

```
Delphi procedure HandCursorClick(ASender: TObject; Element : IElement;Hit :
ShowHandCursorOnEnum;Key : OleVariant);
begin
end;
```

```
Delphi 8 (.NET only) procedure HandCursorClick(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_HandCursorClickEvent);
begin
end;
```

Powe... begin event HandCursorClick(oleobject Element,long Hit,any Key)  
end event HandCursorClick

VB.NET Private Sub HandCursorClick(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_HandCursorClickEvent) Handles  
HandCursorClick  
End Sub

VB6 Private Sub HandCursorClick(ByVal Element As EXSURFACELibCtl.IElement,ByVal  
Hit As EXSURFACELibCtl.ShowHandCursorOnEnum,ByVal Key As Variant)  
End Sub

VBA Private Sub HandCursorClick(ByVal Element As Object,ByVal Hit As Long,ByVal Key  
As Variant)  
End Sub

VFP LPARAMETERS Element,Hit,Key

Xbas... PROCEDURE OnHandCursorClick(oSurface,Element,Hit,Key)  
RETURN

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

Java... <SCRIPT EVENT="HandCursorClick(Element,Hit,Key)" LANGUAGE="JScript">  
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">  
Function HandCursorClick(Element,Hit,Key)  
End Function  
</SCRIPT>

Visual  
Data... Procedure OnComHandCursorClick Variant IElement  
OLEShowHandCursorOnEnum IHit Variant IKey  
Forward Send OnComHandCursorClick IElement IHit IKey  
End\_Procedure

METHOD OCX\_HandCursorClick(Element,Hit,Key) CLASS MainDialog  
RETURN NIL

```
X++ void onEvent_HandCursorClick(COM _Element,int _Hit,COMVariant _Key)
{
}
```

```
XBasic function HandCursorClick as v (Element as OLE::Exontrol.Surface.1::IElement,Hit as
OLE::Exontrol.Surface.1::ShowHandCursorOnEnum,Key as A)
end function
```

```
dBASE function nativeObject_HandCursorClick(Element,Hit,Key)
return
```

The following samples shows how you can handle clicking an icon or a picture of the element:

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

**' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
Private Sub Surface1_HandCursorClick(ByVal Element As Object,ByVal Hit As
Long,ByVal Key As Variant)
    With Surface1
        Debug.Print( Key )
    End With
End Sub
```

```
With Surface1
    .Images
```

```
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrkltl0vr
& _
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _
```

```

"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vI8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="
.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
With .Elements.Add("Caption")
.Pictures = "1,2/pic1/pic2"
.PicturesAlign = 33
.ShowHandCursorOn = 771 '
ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or
ShowHandCursorOnEnum.exShowHandCursorPictures Or
ShowHandCursorOnEnum.exShowHandCursorIcon Or
ShowHandCursorOnEnum.exShowHandCursorPicture
.CaptionAlign = 1
End With
End With

```

## VB6

**' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

Private Sub Surface1_HandCursorClick(ByVal Element As
EXSURFACELibCtl.IElement, ByVal Hit As
EXSURFACELibCtl.ShowHandCursorOnEnum, ByVal Key As Variant)
With Surface1
Debug.Print( Key )
End With
End Sub

```

```

With Surface1
.Images
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrIktl0vr
& _
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _

```



```

"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vI8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="
.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
With .Elements.Add("Caption")
    .Pictures = "1,2/pic1/pic2"
    .PicturesAlign = exBottomCenter
    .ShowHandCursorOn =
ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or
ShowHandCursorOnEnum.exShowHandCursorPictures Or
ShowHandCursorOnEnum.exShowHandCursorIcon Or
ShowHandCursorOnEnum.exShowHandCursorPicture
    .CaptionAlign = exTopCenter
End With
End With

```

## VB.NET

**' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

Private Sub Exsurface1_HandCursorClick(ByVal sender As System.Object,ByVal
Element As exontrol.EXSURFACELib.Element,ByVal Hit As
exontrol.EXSURFACELib.ShowHandCursorOnEnum,ByVal Key As Object) Handles
Exsurface1.HandCursorClick

```

```

    With Exsurface1
        Debug.Print( Key )
    End With
End Sub

```

```

With Exsurface1

```

```

.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oI
& _
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m

```

```

& _
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=")
.set_HTMLPicture("pic1","c:\exontrol\images\zipdisk.gif")
.set_HTMLPicture("pic2","c:\exontrol\images\auction.gif")
With .Elements.Add("Caption")
    .Pictures = "1,2/pic1/pic2"
    .PicturesAlign =
exontrol.EXSURFACELib.ContentAlignmentEnum.exBottomCenter
    .ShowHandCursorOn =
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures
Or exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures Or
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon Or
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture
    .CaptionAlign = exontrol.EXSURFACELib.ContentAlignmentEnum.exTopCenter
End With
End With

```

## VB.NET for /COM

**' HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

Private Sub AxSurface1_HandCursorClick(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib.ISurfaceEvents_HandCursorClickEvent) Handles

```

```

AxSurface1.HandCursorClick

```

```

    With AxSurface1

```

```

        Debug.Print( e.key )

```

```

    End With

```

```

End Sub

```

```

With AxSurface1

```

```

.Images("gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oI

```

```

& _

```

```

"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl

```

```

& _

```

```

"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vI8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=")
.set_HTMLPicture("pic1","c:\exontrol\images\zipdisk.gif")
.set_HTMLPicture("pic2","c:\exontrol\images\auction.gif")
With .Elements.Add("Caption")
    .Pictures = "1,2/pic1/pic2"
    .PicturesAlign = EXSURFACELib.ContentAlignmentEnum.exBottomCenter
    .ShowHandCursorOn =
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures Or
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon Or
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture
    .CaptionAlign = EXSURFACELib.ContentAlignmentEnum.exTopCenter
End With
End With

```

**C++**

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

void OnHandCursorClickSurface1(LPDISPATCH Element,long Hit,VARIANT Key)
{
    /*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXSURFACELib' for the library: 'ExSurface 1.0 Control
Library'
    #import <ExSurface.dll>
    using namespace EXSURFACELib;
    */
    EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
>GetControlUnknown();
    OutputDebugStringW( L"Key" );
}

```

```

EXSURFACELib::ISurfacePtr spSurface1 = GetDlgItem(IDC_SURFACE1)-
> GetControlUnknown();
spSurface1-
> Images(_bstr_t("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIA
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="));
spSurface1->PutHTMLPicture(L"pic1","c:\\exontrol\\images\\zipdisk.gif");
spSurface1->PutHTMLPicture(L"pic2","c:\\exontrol\\images\\auction.gif");
EXSURFACELib::IElementPtr var_Element = spSurface1->GetElements()-
> Add("Caption",vtMissing,vtMissing,vtMissing,vtMissing,vtMissing);
var_Element->PutPictures(L"1,2/pic1/pic2");
var_Element->PutPicturesAlign(EXSURFACELib::exBottomCenter);
var_Element-
> PutShowHandCursorOn(EXSURFACELib::ShowHandCursorOnEnum(EXSURFACELib::ex
| EXSURFACELib::exShowHandCursorPictures |
EXSURFACELib::exShowHandCursorIcon | EXSURFACELib::exShowHandCursorPicture));
var_Element->PutCaptionAlign(EXSURFACELib::exTopCenter);

```

## C++ Builder

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

void __fastcall TForm1::Surface1HandCursorClick(TObject
*Sender,Exsurfacelib_tlb::IElement
*Element,Exsurfacelib_tlb::ShowHandCursorOnEnum Hit,Variant Key)
{
    OutputDebugString( L"Key" );
}

```

Surface1-

```

> Images(TVariant(String("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIE
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="));
Surface1->HTMLPicture[L"pic1"] = TVariant("c:\\exontrol\\images\\zipdisk.gif");
Surface1->HTMLPicture[L"pic2"] = TVariant("c:\\exontrol\\images\\auction.gif");
Exsurfacelib_tlb::IElementPtr var_Element = Surface1->Elements-
>Add(TVariant("Caption"),TNoParam(),TNoParam(),TNoParam(),TNoParam(),TNoParam(

var_Element->Pictures = L"1,2/pic1/pic2";
var_Element->PicturesAlign =
Exsurfacelib_tlb::ContentAlignmentEnum::exBottomCenter;
var_Element->ShowHandCursorOn =
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorExtraPictures |
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorPictures |
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorIcon |
Exsurfacelib_tlb::ShowHandCursorOnEnum::exShowHandCursorPicture;
var_Element->CaptionAlign =
Exsurfacelib_tlb::ContentAlignmentEnum::exTopCenter;

```

C#

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

private void exsurface1_HandCursorClick(object
sender,exontrol.EXSURFACELib.Element
Element,exontrol.EXSURFACELib.ShowHandCursorOnEnum Hit,object Key)
{
    System.Diagnostics.Debug.Print( Key.ToString() );
}
//this.exsurface1.HandCursorClick += new

```

## exontrol.EXSURFACELib.exg2antt.HandCursorClickEventHandler(this.exsurface1

```
exsurface1.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaA  
+  
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
+  
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
+  
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
+  
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");  
exsurface1.set_HTMLPicture("pic1","c:\\exontrol\\images\\zipdisk.gif");  
exsurface1.set_HTMLPicture("pic2","c:\\exontrol\\images\\auction.gif");  
exontrol.EXSURFACELib.Element var_Element =  
exsurface1.Elements.Add("Caption",null,null,null,null,null);  
    var_Element.Pictures = "1,2/pic1/pic2";  
    var_Element.PicturesAlign =  
exontrol.EXSURFACELib.ContentAlignmentEnum.exBottomCenter;  
    var_Element.ShowHandCursorOn =  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures |  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures |  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon |  
exontrol.EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture;  
    var_Element.CaptionAlign =  
exontrol.EXSURFACELib.ContentAlignmentEnum.exTopCenter;
```

## JavaScript

```
<SCRIPT FOR="Surface1" EVENT="HandCursorClick(Element,Hit,Key)"  
LANGUAGE="JScript">  
    alert( Key );  
</SCRIPT>  
  
<OBJECT classid="clsid:AC1DF7F4-0919-4364-8167-2F9B5155EA4B"  
id="Surface1"> </OBJECT>
```

```
<SCRIPT LANGUAGE="JScript">
```

```
Surface1.Images("gBJJgBAIDAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIA/
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");
Surface1.HTMLPicture("pic1") = "c:\\exontrol\\images\\zipdisk.gif";
Surface1.HTMLPicture("pic2") = "c:\\exontrol\\images\\auction.gif";
var var_Element = Surface1.Elements.Add("Caption",null,null,null,null,null);
var_Element.Pictures = "1,2/pic1/pic2";
var_Element.PicturesAlign = 33;
var_Element.ShowHandCursorOn = 771;
var_Element.CaptionAlign = 1;
</SCRIPT>
```

## C# for /COM

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
private void axSurface1_HandCursorClick(object sender,
AxEXSURFACELib._ISurfaceEvents_HandCursorClickEvent e)
{
    System.Diagnostics.Debug.Print( e.key.ToString() );
}
```

**//this.axSurface1.HandCursorClick += new**

**AxEXSURFACELib.\_ISurfaceEvents\_HandCursorClickEventHandler(this.axSurface1**

```

axSurface1.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAJEZFEaIEaEEa/
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");
axSurface1.set_HTMLPicture("pic1","c:\\exontrol\\images\\zipdisk.gif");
axSurface1.set_HTMLPicture("pic2","c:\\exontrol\\images\\auction.gif");
EXSURFACELib.Element var_Element =
axSurface1.Elements.Add("Caption",null,null,null,null,null);
    var_Element.Pictures = "1,2/pic1/pic2";
    var_Element.PicturesAlign =
EXSURFACELib.ContentAlignmentEnum.exBottomCenter;
    var_Element.ShowHandCursorOn =
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures |
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures |
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon |
EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture;
    var_Element.CaptionAlign = EXSURFACELib.ContentAlignmentEnum.exTopCenter;

```

## X++ (Dynamics Ax 2009)

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

void onEvent_HandCursorClick(COM _Element,int _Hit,COMVariant _Key)
{
    ;
    print( _Key );
}

public void init()
{
    COM com_Element;

```



```

anytype var_Element;
str var_s;
;

super();

var_s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

var_s = var_s +
"oFBoVDoIFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu'

var_s = var_s +
"wGBwWDwmFw2HxGJxWLxmNx0xiFdYOTth8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+mC

var_s = var_s +
"3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeBC

var_s = var_s +
"AOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=";
exsurface1.Images(COMVariant::createFromStr(var_s));
exsurface1.HTMLPicture("pic1","c:\\exontrol\\images\\zipdisk.gif");
exsurface1.HTMLPicture("pic2","c:\\exontrol\\images\\auction.gif");
var_Element = COM::createFromObject(exsurface1.Elements()).Add("Caption");
com_Element = var_Element;
com_Element.Pictures("1,2/pic1/pic2");
com_Element.PicturesAlign(33/*exBottomCenter*/);
com_Element.ShowHandCursorOn(771/*exShowHandCursorExtraPictures |
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture*/);
com_Element.CaptionAlign(1/*exTopCenter*/);
}

```

## Delphi 8 (.NET only)

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```

procedure TForm1.AxSurface1_HandCursorClick(sender: System.Object; e:

```

```

AxEXSURFACELib._ISurfaceEvents_HandCursorClickEvent);
begin
  with AxSurface1 do
    begin
      OutputDebugString( e.key );
    end
  end;

with AxSurface1 do
begin

Images('gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIL
+
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu\
+
'wGBwWDwmFw2HxGJxWLxmNx0xiFdYOTth8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0
+
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0
+
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=');
set_HTMLPicture('pic1','c:\exontrol\images\zipdisk.gif');
set_HTMLPicture('pic2','c:\exontrol\images\auction.gif');
with Elements.Add('Caption',Nil,Nil,Nil,Nil,Nil) do
begin
  Pictures := '1,2/pic1/pic2';
  PicturesAlign := EXSURFACELib.ContentAlignmentEnum.exBottomCenter;
  ShowHandCursorOn :=
Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorExtraPictures)
Or Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPictures) Or
Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorIcon) Or
Integer(EXSURFACELib.ShowHandCursorOnEnum.exShowHandCursorPicture);
  CaptionAlign := EXSURFACELib.ContentAlignmentEnum.exTopCenter;
end;
end

```

## Delphi (standard)

**// HandCursorClick event - The uses clicks a part of the element that shows the had cursor.**

```
procedure TForm1.Surface1HandCursorClick(ASender: TObject; Element : IElement;Hit
: ShowHandCursorOnEnum;Key : OleVariant);
begin
  with Surface1 do
  begin
    OutputDebugString( Key );
  end
end;

with Surface1 do
begin

Images('gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIL
+
'oFBoVDolFo1HpFJpVLplNp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu\
+
'wGBwWDwmFw2HxGJxWLxmNx0xiFdyOTh8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0
+
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0
+
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=');
HTMLPicture['pic1'] := 'c:\exontrol\images\zipdisk.gif';
HTMLPicture['pic2'] := 'c:\exontrol\images\auction.gif';
with Elements.Add('Caption',Null,Null,Null,Null,Null) do
begin
  Pictures := '1,2/pic1/pic2';
  PicturesAlign := EXSURFACELib_TLB.exBottomCenter;
  ShowHandCursorOn :=
Integer(EXSURFACELib_TLB.exShowHandCursorExtraPictures) Or
Integer(EXSURFACELib_TLB.exShowHandCursorPictures) Or
```

```

Integer(EXSURFACELib_TLB.exShowHandCursorIcon) Or
Integer(EXSURFACELib_TLB.exShowHandCursorPicture);
    CaptionAlign := EXSURFACELib_TLB.exTopCenter;
end;
end

```

## VFP

```

*** HandCursorClick event - The uses clicks a part of the element that shows the had
cursor. ***
LPARAMETERS Element,Hit,Key
    with thisform.Surface1
        DEBUGOUT( Key )
    endwith

with thisform.Surface1
    var_s =
" gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrkltl0vr

    var_s = var_s +
" oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu'

    var_s = var_s +
" wGBwWDwmFw2HxGJxWLxmNx0xiFdyOTth8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+mC

    var_s = var_s +
" 3GO4NV3WeyvD2XJ5XL5nN51aiw+IfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeBC

    var_s = var_s +
" AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA="
    .Images(var_s)
    .Object.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
    .Object.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
    with .Elements.Add("Caption")
        .Pictures = "1,2/pic1/pic2"
        .PicturesAlign = 33
        .ShowHandCursorOn = 771 &&

```

```
ShowHandCursorOnEnum.exShowHandCursorExtraPictures Or
ShowHandCursorOnEnum.exShowHandCursorPictures Or
ShowHandCursorOnEnum.exShowHandCursorIcon Or
ShowHandCursorOnEnum.exShowHandCursorPicture
    .CaptionAlign = 1
endwith
endwith
```

## dBASE Plus

```
/*
with (this.ACTIVEX1.nativeObject)
    HandCursorClick = class::nativeObject_HandCursorClick
endwith
*/
// The uses clicks a part of the element that shows the had cursor.
function nativeObject_HandCursorClick(Element,Hit,Key)
    local oSurface
    oSurface = form.Activex1.nativeObject
    ? Str(Key)
return

local oSurface,var_Element

oSurface = form.Activex1.nativeObject
oSurface.Images("gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAI/

oSurface.Template = [HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"] //
oSurface.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
oSurface.Template = [HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"] //
oSurface.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
var_Element = oSurface.Elements.Add("Caption")
    var_Element.Pictures = "1,2/pic1/pic2"
    var_Element.PicturesAlign = 33
    var_Element.ShowHandCursorOn = 771 /*exShowHandCursorExtraPictures |
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture*/
    var_Element.CaptionAlign = 1
```

## XBasic (Alpha Five)

```
' The uses clicks a part of the element that shows the had cursor.
function HandCursorClick as v (Element as OLE::Exontrol.Surface.1::IElement,Hit as
OLE::Exontrol.Surface.1::ShowHandCursorOnEnum,Key as A)
    Dim oSurface as P
    oSurface = topparent:CONTROL_ACTIVEX1.activex
    ? Key
end function

Dim oSurface as P
Dim var_Element as P

oSurface = topparent:CONTROL_ACTIVEX1.activex
oSurface.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEalEaEEaAlA/

oSurface.Template = "HTMLPicture(\"pic1\") = \"c:\exontrol\images\zipdisk.gif\""'
oSurface.HTMLPicture("pic1") = "c:\exontrol\images\zipdisk.gif"
oSurface.Template = "HTMLPicture(\"pic2\") = \"c:\exontrol\images\auction.gif\""'
oSurface.HTMLPicture("pic2") = "c:\exontrol\images\auction.gif"
var_Element = oSurface.Elements.Add("Caption")
    var_Element.Pictures = "1,2/pic1/pic2"
    var_Element.PicturesAlign = 33
    var_Element.ShowHandCursorOn = 771 'exShowHandCursorExtraPictures +
exShowHandCursorPictures + exShowHandCursorIcon +
exShowHandCursorPicture
    var_Element.CaptionAlign = 1
```

## Visual Objects

```
METHOD OCX_Exontrol1HandCursorClick(Element,Hit,Key) CLASS MainDialog
    // HandCursorClick event - The uses clicks a part of the element that shows
the had cursor.
    OutputDebugString(String2Psz( AsString(Key) ))
RETURN NIL
```

```
local var_Element as IElement
```

```
oDCOCX_Exontrol1:Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFE
```

```
oDCOCX_Exontrol1:[HTMLPicture,"pic1"] := "c:\exontrol\images\zipdisk.gif"
```

```
oDCOCX_Exontrol1:[HTMLPicture,"pic2"] := "c:\exontrol\images\auction.gif"
```

```
var_Element := oDCOCX_Exontrol1:Elements.Add("Caption",nil,nil,nil,nil,nil)
```

```
var_Element:Pictures := "1,2/pic1/pic2"
```

```
var_Element:PicturesAlign := exBottomCenter
```

```
var_Element:ShowHandCursorOn := exShowHandCursorExtraPictures |
```

```
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture
```

```
var_Element:CaptionAlign := exTopCenter
```

## PowerBuilder

```
/*begin event HandCursorClick(oleobject Element,long Hit,any Key) - The uses clicks a  
part of the element that shows the had cursor.*/
```

```
/*
```

```
OleObject oSurface
```

```
oSurface = ole_1.Object
```

```
MessageBox("Information",string( String(Key) ))
```

```
*/
```

```
/*end event HandCursorClick*/
```

```
OleObject oSurface,var_Element
```

```
oSurface = ole_1.Object
```

```
oSurface.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAI/
```

```
oSurface.HTMLPicture("pic1","c:\exontrol\images\zipdisk.gif")
```

```
oSurface.HTMLPicture("pic2","c:\exontrol\images\auction.gif")
```

```
var_Element = oSurface.Elements.Add("Caption")
```

```
var_Element.Pictures = "1,2/pic1/pic2"
```

```
var_Element.PicturesAlign = 33
```

```
var_Element.ShowHandCursorOn = 771 /*exShowHandCursorExtraPictures |
```

```
exShowHandCursorPictures | exShowHandCursorIcon | exShowHandCursorPicture*/  
var_Element.CaptionAlign = 1
```

## Visual DataFlex

**// The uses clicks a part of the element that shows the had cursor.**

```
Procedure OnComHandCursorClick Variant IIElement OLEShowHandCursorOnEnum  
IIEHit Variant IIKey
```

```
    Forward Send OnComHandCursorClick IIElement IIEHit IIKey
```

```
    ShowIn IIKey
```

```
End_Procedure
```

```
Procedure OnCreate
```

```
    Forward Send OnCreate
```

```
    Send ComImages
```

```
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr
```

```
Set ComHTMLPicture "pic1" to "c:\exontrol\images\zipdisk.gif"
```

```
Set ComHTMLPicture "pic2" to "c:\exontrol\images\auction.gif"
```

```
Variant voElements
```

```
Get ComElements to voElements
```

```
Handle hoElements
```

```
Get Create (RefClass(cComElements)) to hoElements
```

```
Set pvComObject of hoElements to voElements
```

```
    Variant voElement
```

```
    Get ComAdd of hoElements "Caption" Nothing Nothing Nothing Nothing
```

```
Nothing to voElement
```

```
    Handle hoElement
```

```
    Get Create (RefClass(cComElement)) to hoElement
```

```
    Set pvComObject of hoElement to voElement
```

```
        Set ComPictures of hoElement to "1,2/pic1/pic2"
```

```
        Set ComPicturesAlign of hoElement to OLEexBottomCenter
```

```
        Set ComShowHandCursorOn of hoElement to
```

```
(OLEexShowHandCursorExtraPictures + OLEexShowHandCursorPictures +  
OLEexShowHandCursorIcon + OLEexShowHandCursorPicture)
```

```
        Set ComCaptionAlign of hoElement to OLEexTopCenter
```



```
Send Destroy to hoElement
Send Destroy to hoElements
End_Procedure
```

## XBase++

```
PROCEDURE OnHandCursorClick(oSurface,Element,Hit,Key)
    DevOut( Transform(Key,"") )
RETURN
```

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

```
PROCEDURE Main
```

```
    LOCAL oForm
```

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

```
    LOCAL oElement
```

```
    LOCAL oSurface
```

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

```
    oForm:drawingArea:clipChildren := .T.
```

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

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oSurface := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oSurface:CLSID := "Exontrol.Surface.1" /*{AC1DF7F4-0919-4364-8167-
2F9B5155EA4B}*/
```

```
    oSurface:create(,, {10,60},{610,370} )
```

```
    oSurface:HandCursorClick := {||Element,Hit,Key|
```

```
OnHandCursorClick(oSurface,Element,Hit,Key)} /*The uses clicks a part of the element
that shows the had cursor.*/
```

```
oSurface:Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAI/
```

```
oSurface:SetProperty("HTMLPicture","pic1","c:\exontrol\images\zipdisk.gif")
```

```
oSurface:SetProperty("HTMLPicture","pic2","c:\exontrol\images\auction.gif")
oElement := oSurface:Elements():Add("Caption")
  oElement:Pictures := "1,2/pic1/pic2"
  oElement:PicturesAlign := 33/*exBottomCenter*/
  oElement:ShowHandCursorOn :=
771/*exShowHandCursorExtraPictures+exShowHandCursorPictures+exShowHandCurs

  oElement:CaptionAlign := 1/*exTopCenter*/

oForm:Show()
DO WHILE nEvent != xbeP_Quit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
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

You can use the [Edit](#) method of the Element object to starts editing the element's [Caption](#) or [ExtraCaption](#).

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,
AxEXSURFACELib._ISurfaceEvents_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:
AxEXSURFACELib._ISurfaceEvents_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
AxEXSURFACELib._ISurfaceEvents_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(oSurface,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)  
{  
}

VBPrivate 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,  
AxEXSURFACELib.\_ISurfaceEvents\_KeyPressEvent e)  
{  
}

C++void OnKeyPress(short FAR\* KeyAscii)  
{  
}

C++ Buildervoid \_\_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:  
AxEXSURFACELib.\_ISurfaceEvents\_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  
AxEXSURFACELib.\_ISurfaceEvents\_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(oSurface,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 Integer KeyAscii  
    Forward Send OnComKeyPress Integer KeyAscii  
End_Procedure
```

Visual  
Objects

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

C++

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

XBasic

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

dBASE

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



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

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

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

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

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

C#	<pre>private void KeyUp(object sender,ref short KeyCode,short Shift) { }</pre>
VB	<pre>Private Sub KeyUp(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal Shift As Short) Handles KeyUp End Sub</pre>

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

C#	<pre>private void KeyUpEvent(object sender, AxEXSURFACELib._ISurfaceEvents_KeyUpEvent e) { }</pre>
C++	<pre>void OnKeyUp(short FAR* KeyCode,short Shift) { }</pre>
C++ Builder	<pre>void __fastcall KeyUp(TObject *Sender,short * KeyCode,short Shift) {</pre>

```
}
```

**Delphi**

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

**Delphi 8  
(.NET  
only)**

```
procedure KeyUpEvent(sender: System.Object; e:  
AxEXSURFACELib._ISurfaceEvents_KeyUpEvent);  
begin  
end;
```

**Powe...**

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

**VB.NET**

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_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(oSurface,KeyCode,Shift)  
RETURN
```

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

**Java...**

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

**VBSc...**

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

</SCRIPT>

Visual  
Data...

```
Procedure OnComKeyUp Short Integer KeyCode Short Integer Shift
    Forward Send OnComKeyUp Integer KeyCode Integer Shift
End_Procedure
```

Visual  
Objects

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

C++

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

XBasic

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

dBASE

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

# event **LayoutEndChanging** (Operation as **LayoutChangingEnum**)

Notifies your application once the control's layout has been changed.

Type	Description
Operation as <a href="#">LayoutChangingEnum</a>	A <a href="#">LayoutChangingEnum</a> expression that specifies the operation that ends.

The [LayoutEndChanging](#) event notifies that the specified operation ends. The [LayoutStartChanging](#) event occurs once the user starts an operation on the surface, like creating / adding a new element to the surface. During the [LayoutStartChanging](#) event, you can call the [CancelLayoutChanging](#) method to cancel the specified operation.

The operations being signaled by the [LayoutStartChanging](#) / [LayoutEndChanging](#) events are:

- **exSurfaceMove**, the user scrolls or moves the surface. The [AllowMoveSurface](#) property specifies the keys combination to allow user to move / scroll the surface.
- **exSurfaceZoom**, the user magnifies or shrinks the surface. The [AllowZoomSurface](#) property specifies the keys combination to allow user to zoom the surface.
- **exSurfaceHome**, the user clicks the Home button on the control's toolbar, so the surface is restored to original position. The [Home](#) method has the same effect.
- **exResizeObject**, the user resizes the object. The [AllowResizeObject](#) property specifies the keys combination to allow user to resize the object.
- **exMoveObject**, the user moves the object. The [AllowMoveObject](#) property specifies the keys combination to allow user to move the object.
- **exSelectObject**, the user clicks the object to get it selected. The [AllowSelectObject](#) property specifies the keys combination to allow user to select the object.
- **exSelectNothing**, the user clicks an empty zone of the surface. The [AllowSelectNothing](#) property specifies the keys combination to allow user to select nothing on the surface.
- **exCreateObject**, the user creates an element on the surface. The [AllowCreateObject](#) property specifies the keys combination to allow user to create elements on the surface.
- **exEditObject**, the user edits the element's caption.
- **exLinkObjects**, the user creates an element on the surface. The [AllowLinkObjects](#) property specifies the keys combination to allow user to link elements on the surface
- **exFocusLink**, the user clicks a link (the focused link is being updated). The [FocusLink](#) property retrieves or changes the current link that is currently focused (selected or active) within the control.
- **exUndo**, An Undo operation is performed (CTR + Z). Occurs only if the control's [AllowUndoRedo](#) property is True.
- **exRedo**, A Redo operation is performed (CTR + Y). Occurs only if the control's

[AllowUndoRedo](#) property is True.

- **exUndoRedoUpdate**, The Undo/Redo queue is updated. Occurs only if the control's [AllowUndoRedo](#) property is True.

For instance, the following events occur when user creates an element on the surface:

- [LayoutStartChanging](#)(exCreateObject), the user clicks on the surface
- [AddElement](#), adds the new element to the Elements collection
- [CreateElement](#), the user ends creating the object
- [LayoutEndChanging](#)(exCreateObject), the user un-clicks the surface

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

```
C# private void LayoutEndChanging(object sender,excontrol.EXSURFACELib.LayoutChangingEnum Operation)
{
}
```

```
VB Private Sub LayoutEndChanging(ByVal sender As System.Object,ByVal Operation As excontrol.EXSURFACELib.LayoutChangingEnum) Handles LayoutEndChanging
End Sub
```

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

```
C# private void LayoutEndChanging(object sender,
AxEXSURFACELib._ISurfaceEvents_LayoutEndChangingEvent e)
{
}
```

```
C++ void OnLayoutEndChanging(long Operation)
{
}
```

```
C++ Builder void __fastcall LayoutEndChanging(TObject
*Sender,Exsurfacelib_tlb::LayoutChangingEnum Operation)
{
}
```

```
Delphi procedure LayoutEndChanging(ASender: TObject; Operation :
LayoutChangingEnum);
```

```
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure LayoutEndChanging(sender: System.Object; e:  
AxEXSURFACELib._ISurfaceEvents_LayoutEndChangingEvent);  
begin  
end;
```

Power...

```
begin event LayoutEndChanging(long Operation)  
end event LayoutEndChanging
```

VB.NET

```
Private Sub LayoutEndChanging(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_LayoutEndChangingEvent) Handles  
LayoutEndChanging  
End Sub
```

VB6

```
Private Sub LayoutEndChanging(ByVal Operation As  
EXSURFACELibCtl.LayoutChangingEnum)  
End Sub
```

VBA

```
Private Sub LayoutEndChanging(ByVal Operation As Long)  
End Sub
```

VFP

```
LPARAMETERS Operation
```

Xbas...

```
PROCEDURE OnLayoutEndChanging(oSurface,Operation)  
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComLayoutEndChanging OLELayoutChangingEnum IIOperation  
    Forward Send OnComLayoutEndChanging IIOperation  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_LayoutEndChanging(Operation) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_LayoutEndChanging(int _Operation)  
{  
}
```

XBasic

```
function LayoutEndChanging as v (Operation as  
OLE::Exontrol.Surface.1::LayoutChangingEnum)  
end function
```

dBASE

```
function nativeObject_LayoutEndChanging(Operation)  
return
```

# event **LayoutStartChanging** (Operation as **LayoutChangingEnum**)

Occurs when the control's layout is about to be changed.

Type	Description
Operation as <a href="#">LayoutChangingEnum</a>	A <b>LayoutChangingEnum</b> expression that specifies the operation that begins.

The **LayoutStartChanging** event occurs once the user starts an operation on the surface, like creating / adding a new element to the surface. During the **LayoutStartChanging** event, you can call the [CancelLayoutChanging](#) method to cancel the specified operation. The [LayoutEndChanging](#) event notifies that the specified operation ends.

The operations being signaled by the **LayoutStartChanging** / [LayoutEndChanging](#) events are:

- **exSurfaceMove**, the user scrolls or moves the surface. The [AllowMoveSurface](#) property specifies the keys combination to allow user to move / scroll the surface.
- **exSurfaceZoom**, the user magnifies or shrinks the surface. The [AllowZoomSurface](#) property specifies the keys combination to allow user to zoom the surface.
- **exSurfaceHome**, the user clicks the Home button on the control's toolbar, so the surface is restored to original position. The [Home](#) method has the same effect.
- **exResizeObject**, the user resizes the object. The [AllowResizeObject](#) property specifies the keys combination to allow user to resize the object.
- **exMoveObject**, the user moves the object. The [AllowMoveObject](#) property specifies the keys combination to allow user to move the object.
- **exSelectObject**, the user clicks the object to get it selected. The [AllowSelectObject](#) property specifies the keys combination to allow user to select the object.
- **exSelectNothing**, the user clicks an empty zone of the surface. The [AllowSelectNothing](#) property specifies the keys combination to allow user to select nothing on the surface.
- **exCreateObject**, the user creates an element on the surface. The [AllowCreateObject](#) property specifies the keys combination to allow user to create elements on the surface.
- **exEditObject**, the user edits the element's caption.
- **exLinkObjects**, the user creates an element on the surface. The [AllowLinkObjects](#) property specifies the keys combination to allow user to link elements on the surface.
- **exFocusLink**, the user clicks a link (the focused link is being updated). The [FocusLink](#) property retrieves or changes the current link that is currently focused (selected or active) within the control.
- **exUndo**, An Undo operation is performed (CTR + Z). Occurs only if the control's [AllowUndoRedo](#) property is True.
- **exRedo**, A Redo operation is performed (CTR + Y). Occurs only if the control's



[AllowUndoRedo](#) property is True.

- **exUndoRedoUpdate**, The Undo/Redo queue is updated. Occurs only if the control's [AllowUndoRedo](#) property is True.

For instance, the following events occur when user creates an element on the surface:

- LayoutStartChanging(exCreateObject), the user clicks on the surface
- [AddElement](#), adds the new element to the Elements collection
- [CreateElement](#), the user ends creating the object
- [LayoutEndChanging](#)(exCreateObject), the user un-clicks the surface

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

```
C# private void LayoutStartChanging(object
sender,excontrol.EXSURFACELib.LayoutChangingEnum Operation)
{
}
```

```
VB Private Sub LayoutStartChanging(ByVal sender As System.Object,ByVal Operation
As excontrol.EXSURFACELib.LayoutChangingEnum) Handles LayoutStartChanging
End Sub
```

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

```
C# private void LayoutStartChanging(object sender,
AxEXSURFACELib._ISurfaceEvents_LayoutStartChangingEvent e)
{
}
```

```
C++ void OnLayoutStartChanging(long Operation)
{
}
```

```
C++ Builder void __fastcall LayoutStartChanging(TObject
*Sender,Exsurfacelib_tlb::LayoutChangingEnum Operation)
{
}
```

```
Delphi procedure LayoutStartChanging(ASender: TObject; Operation :
LayoutChangingEnum);
```

```
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure LayoutStartChanging(sender: System.Object; e:  
AxEXSURFACELib._ISurfaceEvents_LayoutStartChangingEvent);  
begin  
end;
```

Power...

```
begin event LayoutStartChanging(long Operation)  
end event LayoutStartChanging
```

VB.NET

```
Private Sub LayoutStartChanging(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_LayoutStartChangingEvent) Handles  
LayoutStartChanging  
End Sub
```

VB6

```
Private Sub LayoutStartChanging(ByVal Operation As  
EXSURFACELibCtl.LayoutChangingEnum)  
End Sub
```

VBA

```
Private Sub LayoutStartChanging(ByVal Operation As Long)  
End Sub
```

VFP

```
LPARAMETERS Operation
```

Xbas...

```
PROCEDURE OnLayoutStartChanging(oSurface,Operation)  
RETURN
```

Syntax for LayoutStartChanging event, **ICOM** version (others), on:

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComLayoutStartChanging OLELayoutChangingEnum llOperation  
    Forward Send OnComLayoutStartChanging llOperation  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_LayoutStartChanging(Operation) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_LayoutStartChanging(int _Operation)  
{  
}
```

XBasic

```
function LayoutStartChanging as v (Operation as  
OLE::Exontrol.Surface.1::LayoutChangingEnum)  
end function
```

dBASE

```
function nativeObject_LayoutStartChanging(Operation)  
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. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#)(-1,-1) property returns the element and the hit-test code from the cursor. The [HandCursorClick](#) event notifies once the user clicks a part of the element ( which shows a hand cursor when the pointer hovers it ).

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,  
AxEXSURFACELib._ISurfaceEvents_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:  
AxEXSURFACELib._ISurfaceEvents_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  
AxEXSURFACELib._ISurfaceEvents_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(oSurface,Button,Shift,X,Y)
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComMouseDown Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
    Forward Send OnComMouseDown IButton IShift IIX IY
End_Procedure
```

Visual  
Objects

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

X++

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

XBasic

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

dBASE

```
function nativeObject_MouseDown(Button,Shift,X,Y)
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.
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. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#)(-1,-1) property returns the element and the hit-test code from the cursor. The [ShowHandCursorOn](#) property specifies the parts of the element that shows the hand cursor when the mouse-pointer hovers the part. The [HitTestFromPoint](#)(-1,-1) property returns the element/hit-test code from the cursor. Use the [LinkFromPoint](#) property to get if there is any link at the specified position.

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

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

VBPrivate Sub MouseEventArgsEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseEventArgsEventEnd Sub

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

C#private void MouseEventArgsEvent(object sender,

```
AxEXSURFACELib._ISurfaceEvents_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: AxEXSURFACELib._ISurfaceEvents_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 AxEXSURFACELib._ISurfaceEvents_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(oSurface,Button,Shift,X,Y)
RETURN
```

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

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

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

```
Visual Data... Procedure OnComMouseMove Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
    Forward Send OnComMouseMove IButton IShift IIX IY
End_Procedure
```

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

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

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

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

The following VB sample determines if the cursor hovers the element's expand/collapse glyphs:

```
Private Sub Surface1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
```

Single)

Dim h As HitTest

Set h = Surface1.HitTestFromPoint(-1, -1)

If Not h Is Nothing Then

    If (h.HitTestCode And exHitTestMask) = exHitTestGlyph Then

        Debug.Print "Expand/Collapse Glyph of " & h.Element.ID

    End If

End If

End Sub

# 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. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#)(-1,-1) property returns the element and the hit-test code from the cursor. The [HandCursorClick](#) event notifies once the user clicks a part of the element ( which shows a hand cursor when the pointer hovers it ).

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,
AxEXSURFACELib._ISurfaceEvents_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:
AxEXSURFACELib._ISurfaceEvents_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
AxEXSURFACELib._ISurfaceEvents_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(oSurface,Button,Shift,X,Y)
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComMouseUp Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
    Forward Send OnComMouseUp IButton IShift IIX IY
End_Procedure
```

Visual  
Objects

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

X++

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

XBasic

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

dBASE

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

# event **OLECompleteDrag** (Effect as Long)

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled

Type	Description
Effect as Long	A long set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another.

The **OLECompleteDrag** event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the effect parameter of the [OLEDragDrop](#) event. Based on this, the source can then determine the appropriate action it needs to take. For example, if the object was moved into the target (exDropEffectMove), the source needs to delete the object from itself after the move. The control supports only manual OLE drag and drop events. In order to enable OLE drag and drop feature into control you have to set the [OLEDropMode](#) and [OLEDrag](#) properties.

The settings for Effect are:

- exOLEDropEffectNone (0), Drop target cannot accept the data, or the drop operation was cancelled
- exOLEDropEffectCopy (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- exOLEDropEffectMove (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

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

```
C# // OLECompleteDrag event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

```
VB // OLECompleteDrag event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

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

```
C# private void OLECompleteDrag(object sender,  
AxEXSURFACELib._ISurfaceEvents_OLECompleteDragEvent e)  
{
```

```
}
```

C++

```
void OnOLECompleteDrag(long Effect)
{
}
```

C++  
Builder

```
void __fastcall OLECompleteDrag(TObject *Sender,long Effect)
{
}
```

Delphi

```
procedure OLECompleteDrag(ASender: TObject; Effect : Integer);
begin
end;
```

Delphi 8  
(.NET  
only)

```
procedure OLECompleteDrag(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_OLECompleteDragEvent);
begin
end;
```

Power...

```
begin event OLECompleteDrag(long Effect)
end event OLECompleteDrag
```

VB.NET

```
Private Sub OLECompleteDrag(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OLECompleteDragEvent) Handles
OLECompleteDrag
End Sub
```

VB6

```
Private Sub OLECompleteDrag(ByVal Effect As Long)
End Sub
```

VBA

```
Private Sub OLECompleteDrag(ByVal Effect As Long)
End Sub
```

VFP

```
LPARAMETERS Effect
```

Xbas...

```
PROCEDURE OnOLECompleteDrag(oSurface,Effect)
RETURN
```

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

Java... <SCRIPT EVENT="OLECompleteDrag(Effect)" LANGUAGE="JScript">  
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">  
Function OLECompleteDrag(Effect)  
End Function  
</SCRIPT>

Visual  
Data... Procedure OnComOLECompleteDrag Integer lEffect  
Forward Send OnComOLECompleteDrag lEffect  
End\_Procedure

Visual  
Objects METHOD OCX\_OLECompleteDrag(Effect) CLASS MainDialog  
RETURN NIL

X++ // OLECompleteDrag event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.

XBasic function OLECompleteDrag as v (Effect as N)  
end function

dBASE function nativeObject\_OLECompleteDrag(Effect)  
return



**event OLEDragDrop (Data as ExDataObject, Effect as Long, Button as Integer, Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS)**

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Type	Description
Data as <a href="#">ExDataObject</a>	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here.
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in bellow.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed
Shift as Integer	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
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.

*In the /NET Assembly, you have to use the DragDrop event as explained here:*

- <https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop>

The OLEDragDrop event is fired when the user has dropped files or clipboard information into the control. Use the [OLEDropMode](#) property on exOLEDropManual to enable OLE drop and drop support.

The settings for Effect are:

- exOLEDropEffectNone (0), Drop target cannot accept the data, or the drop operation was cancelled
- exOLEDropEffectCopy (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- exOLEDropEffectMove (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

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

```
C# // OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
    DragDrop ... events.
```

```
VB // OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
    DragDrop ... events.
```

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

```
C# private void OLEDragDrop(object sender,
    AxEXSURFACELib._ISurfaceEvents_OLEDragDropEvent e)
    {
    }
```

```
C++ void OnOLEDragDrop(LPDISPATCH Data,long FAR* Effect,short Button,short
    Shift,long X,long Y)
    {
    }
```

```
C++ Builder void __fastcall OLEDragDrop(TObject *Sender,Exsurfacelib_tlb::IExDataObject
    *Data,long * Effect,short Button,short Shift,int X,int Y)
    {
    }
```

**Delphi** procedure OLEDragDrop(ASender: TObject; Data : IExDataObject;var Effect : Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer);  
begin  
end;

**Delphi 8  
(.NET  
only)** procedure OLEDragDrop(sender: System.Object; e:  
AxEXSURFACELib.\_ISurfaceEvents\_OLEDragDropEvent);  
begin  
end;

**Powe...** begin event OLEDragDrop(oleobject Data,long Effect,integer Button,integer  
Shift,long X,long Y)  
end event OLEDragDrop

**VB.NET** Private Sub OLEDragDrop(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_OLEDragDropEvent) Handles OLEDragDrop  
End Sub

**VB6** Private Sub OLEDragDrop(ByVal Data As EXSURFACELibCtl.IExDataObject,Effect As  
Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As  
Single)  
End Sub

**VBA** Private Sub OLEDragDrop(ByVal Data As Object,Effect As Long,ByVal Button As  
Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)  
End Sub

**VFP** LPARAMETERS Data,Effect,Button,Shift,X,Y

**Xbas...** PROCEDURE OnOLEDragDrop(oSurface,Data,Effect,Button,Shift,X,Y)  
RETURN

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

**Java...** <SCRIPT EVENT="OLEDragDrop(Data,Effect,Button,Shift,X,Y)"  
LANGUAGE="JScript">  
</SCRIPT>

**VBSc...**

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

**Visual  
Data...**

```
Procedure OnComOLEDragDrop Variant IIData Integer IIEffect Short IIButton  
Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS IIY  
    Forward Send OnComOLEDragDrop IIData IIEffect IIButton IIShift IIX IIY  
End_Procedure
```

**Visual  
Objects**

```
METHOD OCX_OLEDragDrop(Data,Effect,Button,Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

**X++**

```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

**XBasic**

```
function OLEDragDrop as v (Data as OLE::Exontrol.Surface.1::IExDataObject,Effect  
as N,Button as N,Shift as N,X as OLE::Exontrol.Surface.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.Surface.1::OLE_YPOS_PIXELS)  
end function
```

**dBASE**

```
function nativeObject_OLEDragDrop(Data,Effect,Button,Shift,X,Y)  
return
```

The following VB sample adds a new item when the user drags a file ( Open the Windows Explorer, click and drag a file to the control ) :

```
Private Sub Surface1_OLEDragDrop(Index As Integer, ByVal Data As  
EXSURFACELibCtl.IExDataObject, Effect As Long, ByVal Button As Integer, ByVal Shift As  
Integer, ByVal X As Single, ByVal Y As Single)  
    If Data.GetFormat(exCFFiles) Then  
        Data.GetData (exCFFiles)  
        Dim strFile As String  
        strFile = Data.Files(0)  
        'Adds a new item to the control  
        Surface1(Index).Visible = False  
        With Surface1(Index)
```

```

.BeginUpdate
    Dim i As HITEM
    i = .Items.AddItem(strFile)
    .Items.EnsureVisibleItem i
.EndUpdate
End With
Surface1(Index).Visible = True
End If
End Sub

```

The following VC sample inserts a child item for each file that user drags:

```

#import <exsurface.dll> rename( "GetItems", "exGetItems" )

#include "Items.h"
void OnOLEDragDropSurface1(LPDISPATCH Data, long FAR* Effect, short Button, short
Shift, long X, long Y)
{
    EXSURFACELib::IExDataObjectPtr spData( Data );
    if ( spData != NULL )
        if ( spData->GetFormat( EXSURFACELib::exCFFiles ) )
        {
            CItems items = m_surface.GetItems();
            // Gets the handle of the item where the files will be inserted
            long c = 0, h = 0, nParentItem = m_surface.GetItemFromPoint( X, Y, &c, &h );
            if ( nParentItem == 0 )
                if ( c != 0 )
                    nParentItem = items.GetCellItem( c );
            EXSURFACELib::IExDataObjectFilesPtr spFiles( spData->Files );
            if ( spFiles->Count > 0 )
            {
                m_surface.BeginUpdate();
                COleVariant vtMissing; vtMissing.vt = VT_ERROR;
                for ( long i = 0; i < spFiles->Count; i++ )
                    items.InsertItem( nParentItem, vtMissing, COleVariant( spFiles->GetItem( i
).operator const char *() ) );
                if ( nParentItem )

```

```

        items.SetExpandItem( nParentItem, TRUE );
        m_surface.EndUpdate();
    }

}

}

```

The #import statement imports definition for the [ExDataObject](#) and [ExDataObjectFiles](#) objects. If the exsurface.dll file is located in another folder than the system folder, the path to the file must be specified. The sample gets the item where the files were dragged and insert all files in that position, as child items, if case.

The following VB.NET sample inserts a child item for each file that user drags:

```

Private Sub AxSurface1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OLEDragDropEvent) Handles AxSurface1.OLEDragDrop
    If e.data.GetFormat(EXSURFACELib.exClipboardFormatEnum.exCFFiles) Then
        If (e.data.Files.Count > 0) Then
            AxSurface1.BeginUpdate()
            With AxSurface1.Items
                Dim iParent As Integer, c As Integer, hit As EXSURFACELib.HitTestInfoEnum
                iParent = AxSurface1.get_ItemFromPoint(e.x, e.y, c, hit)
                If iParent = 0 Then
                    If Not c = 0 Then
                        iParent = .CellItem(c)
                    End If
                End If
                Dim i As Long
                For i = 0 To e.data.Files.Count - 1
                    .InsertItem(iParent, , e.data.Files(i))
                Next
                If Not (iParent = 0) Then
                    .ExpandItem(iParent) = True
                End If
            End With
            AxSurface1.EndUpdate()
        End If
    End If
End Sub

```

The following C# sample inserts a child item for each file that user drags:

```
private void axSurface1_OLEDragDrop(object sender,
AxEXSURFACELib._ISurfaceEvents_OLEDragDropEvent e)
{
    if ( e.data.GetFormat(
Convert.ToInt16(EXSURFACELib.exClipboardFormatEnum.exCFFiles) ) )
        if ( e.data.Files.Count > 0 )
        {
            EXSURFACELib.HitTestInfoEnum hit;
            int c = 0, iParent = axSurface1.get_ItemFromPoint( e.x, e.y, out c, out hit );
            if ( iParent == 0 )
                if ( c != 0 )
                    iParent = axSurface1.Items.get_CellItem( c );

            axSurface1.BeginUpdate();
            for ( int i = 0; i < e.data.Files.Count; i++ )
                axSurface1.Items.InsertItem( iParent, "", e.data.Files[i].ToString() );
            if ( iParent != 0 )
                axSurface1.Items.set_ExpandItem( iParent, true );
            axSurface1.EndUpdate();
        }
}
```

The following VFP sample inserts a child item for each file that user drags:

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

local c, hit, iParent
c = 0
hit = 0
if ( data.GetFormat( 15 ) ) && exCFFiles
    if ( data.Files.Count() > 0 )
        with thisform.Surface1.Items
            iParent = thisform.Surface1.ItemFromPoint( x, y, @c, @hit )
```

```
thisform.Surface1.BeginUpdate()
for i = 0 to data.files.Count() - 1
    .InsertItem( iParent, "", data.files(i) )
next
if ( iParent != 0 )
    .DefaultItem = iParent
    .ExpandItem( 0 ) = .t.
endif
thisform.Surface1.EndUpdate()
endwith
endif
endif
```



**event OLEDragOver (Data as ExDataObject, Effect as Long, Button as Integer, Shift as Integer, X as OLE\_XPOS\_PIXELS, Y as OLE\_YPOS\_PIXELS, State as Integer)**

Occurs when one component is dragged over another.

Type	Description
Data as <a href="#">ExDataObject</a>	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed bellow.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
Shift as Integer	These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
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.
State as Integer	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed bellow.

The settings for effect are:

- exOLEDropEffectNone (0), Drop target cannot accept the data, or the drop operation was cancelled
- exOLEDropEffectCopy (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- exOLEDropEffectMove (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

The settings for state are:

- exOLEDragEnter (0), Source component is being dragged within the range of a target.
- exOLEDragLeave (1), Source component is being dragged out of the range of a target.
- exOLEOLEDragOver (2), Source component has moved from one position in the target to another.

Note If the state parameter is 1, indicating that the mouse pointer has left the target, then the x and y parameters will contain zeros.

The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:

If Effect = exOLEDropEffectCopy...

Instead, the source component should mask for the value or values being sought, such as this:

If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...

-or-

If (Effect And exOLEDropEffectCopy)...

This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

The control supports only manual OLE drag and drop events.

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

```
C# // OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

```
VB // OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

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

C#	<pre>private void OLEDragOver(object sender, AxEXSURFACELib._ISurfaceEvents_OLEDragOverEvent e) { }</pre>
C++	<pre>void OnOLEDragOver(LPDISPATCH Data,long FAR* Effect,short Button,short Shift,long X,long Y,short State) { }</pre>
C++ Builder	<pre>void __fastcall OLEDragOver(TObject *Sender,Exsurfacelib_tlb::IExDataObject *Data,long * Effect,short Button,short Shift,int X,int Y,short State) { }</pre>
Delphi	<pre>procedure OLEDragOver(ASender: TObject; Data : IExDataObject;var Effect : Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer;State : Smallint); begin end;</pre>
Delphi 8 (.NET only)	<pre>procedure OLEDragOver(sender: System.Object; e: AxEXSURFACELib._ISurfaceEvents_OLEDragOverEvent); begin end;</pre>
Powe...	<pre>begin event OLEDragOver(oleobject Data,long Effect,integer Button,integer Shift,long X,long Y,integer State) end event OLEDragOver</pre>
VB.NET	<pre>Private Sub OLEDragOver(ByVal sender As System.Object, ByVal e As AxEXSURFACELib._ISurfaceEvents_OLEDragOverEvent) Handles OLEDragOver End Sub</pre>
VB6	<pre>Private Sub OLEDragOver(ByVal Data As EXSURFACELibCtl.IExDataObject,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As Single,ByVal State As Integer)</pre>

End Sub

VBA

```
Private Sub OLEDragOver(ByVal Data As Object,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long,ByVal State As Integer)
End Sub
```

VFP

```
LPARAMETERS Data,Effect,Button,Shift,X,Y,State
```

Xbas...

```
PROCEDURE OnOLEDragOver(oSurface,Data,Effect,Button,Shift,X,Y,State)
RETURN
```

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

Java...

```
<SCRIPT EVENT="OLEDragOver(Data,Effect,Button,Shift,X,Y,State)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
End Function
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComOLEDragOver Variant IIData Integer IIEffect Short IIButton Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS IIY Short IIShift
Forward Send OnComOLEDragOver IIData IIEffect IIButton IIShift IIX IIY IIShift
End_Procedure
```

Visual  
Objects

```
METHOD OCX_OLEDragOver(Data,Effect,Button,Shift,X,Y,State) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

XBasic

```
function OLEDragOver as v (Data as OLE::Exontrol.Surface.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.Surface.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Surface.1::OLE_YPOS_PIXELS,State as N)
```

end function

dBASE

```
function nativeObject_OLEDragOver(Data,Effect,Button,Shift,X,Y,State)  
return
```

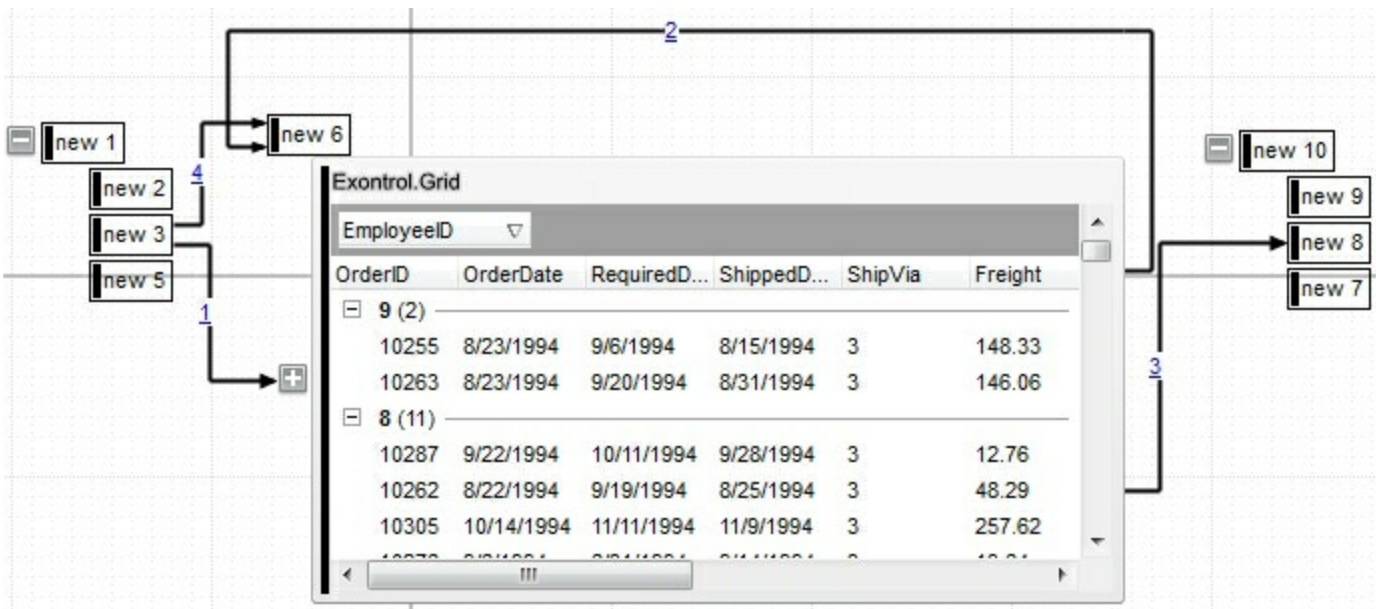
# event OleEvent (Element as Element, Ev as OleEvent)

Occurs once an inside control fires an event.

Type	Description
Element as <a href="#">Element</a>	An Element object that hosts the ActiveX control.
Ev as <a href="#">OleEvent</a>	A OleEvent object that contains information about the event.

The ExSurface component supports ActiveX hosting. The [Type](#) property on exElementHostControl, specifies that the element hosts an inside control. The [Control](#) property specifies the program identifier of the control to be hosted. In case you are inserting a runtime-licensed control you must specify the [License](#) property, prior calling the Control property. The [Object](#) property specifies the object being hosted ( a reference to the inside ActiveX control ). The OleEvent event occurs once an inner control fires an event. Use the [ToString](#) property of the OleEvent object to display general information about the fired event.

The following screen shot shows the Excontrol.Grid on the surface:



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

C#	<pre>private void OleEvent(object sender,excontrol.EXSURFACELib.Element Element,excontrol.EXSURFACELib.OleEvent Ev) { }</pre>
VB	<pre>Private Sub OleEvent(ByVal sender As System.Object,ByVal Element As excontrol.EXSURFACELib.Element,ByVal Ev As excontrol.EXSURFACELib.OleEvent)</pre>

```
Handles OleEvent
End Sub
```

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

```
C# private void OleEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_OleEventEvent e)
{
}
```

```
C++ void OnOleEvent(LPDISPATCH Element,LPDISPATCH Ev)
{
}
```

```
C++ Builder void __fastcall OleEvent(TObject *Sender,Exsurfacelib_tlb::IElement
*Element,Exsurfacelib_tlb::IOleEvent *Ev)
{
}
```

```
Delphi procedure OleEvent(ASender: TObject; Element : IElement;Ev : IOleEvent);
begin
end;
```

```
Delphi 8 (.NET only) procedure OleEvent(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_OleEventEvent);
begin
end;
```

```
Powe... begin event OleEvent(oleobject Element,oleobject Ev)
end event OleEvent
```

```
VB.NET Private Sub OleEvent(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OleEventEvent) Handles OleEvent
End Sub
```

```
VB6 Private Sub OleEvent(ByVal Element As EXSURFACELibCtl.IElement,ByVal Ev As
EXSURFACELibCtl.IOleEvent)
End Sub
```

**VBA** Private Sub OleEvent(ByVal Element As Object,ByVal Ev As Object)  
End Sub

**VFP** LPARAMETERS Element,Ev

**Xbas...** PROCEDURE OnOleEvent(oSurface,Element,Ev)  
RETURN

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

**Java...** <SCRIPT EVENT="OleEvent(Element,Ev)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function OleEvent(Element,Ev)  
End Function  
</SCRIPT>

**Visual Data...** Procedure OnComOleEvent Variant IElement Variant IIEv  
Forward Send OnComOleEvent IElement IIEv  
End\_Procedure

**Visual Objects** METHOD OCX\_OleEvent(Element,Ev) CLASS MainDialog  
RETURN NIL

**X++** void onEvent\_OleEvent(COM \_Element,COM \_Ev)  
{  
}

**XBasic** function OleEvent as v (Element as OLE::Exontrol.Surface.1::IElement,Ev as  
OLE::Exontrol.Surface.1::IOleEvent)  
end function

**dBASE** function nativeObject\_OleEvent(Element,Ev)  
return



The following VB sample adds a command button:

```
With Surface1
  With .Elements
    With .Add("activex hosting")
      .Type = exElementHostControl
      .Control = "Forms.CommandButton.1"
    End With
  End With
End With
```

The following sample displays information about fired event:

```
Private Sub Surface1_OleEvent(ByVal Element As EXSURFACELibCtl.IElement, ByVal Ev As EXSURFACELibCtl.IOleEvent)
  Debug.Print Ev.ToString()
End Sub
```

# event OLEGiveFeedback (Effect as Long, DefaultCursors as Boolean)

Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.

Type	Description
Effect as Long	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed bellow.
DefaultCursors as Boolean	Boolean value that determines whether to use the default mouse cursor, or to use a user-defined mouse cursor.True (default) = use default mouse cursor.False = do not use default cursor. Mouse cursor must be set with the MousePointer property of the Screen object

The settings for Effect are:

- exOLEDropEffectNone (0), Drop target cannot accept the data, or the drop operation was cancelled
- exOLEDropEffectCopy (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- exOLEDropEffectMove (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

If there is no code in the OLEGiveFeedback event, or if the defaultcursors parameter is set to True, the mouse cursor will be set to the default cursor provided by the control. The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:

If Effect = exOLEDropEffectCopy...

Instead, the source component should mask for the value or values being sought, such as this:

If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...

-or-

If (Effect And exOLEDropEffectCopy)...

This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

The control supports only manual OLE drag and drop events.

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

```
C# // OLEGiveFeedback event is not supported. Use the
    DragEnter,DragLeave,DragOver, DragDrop ... events.
```

```
VB // OLEGiveFeedback event is not supported. Use the
    DragEnter,DragLeave,DragOver, DragDrop ... events.
```

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

```
C# private void OLEGiveFeedback(object sender,
    AxEXSURFACELib._ISurfaceEvents_OLEGiveFeedbackEvent e)
    {
    }
```

```
C++ void OnOLEGiveFeedback(long Effect,BOOL FAR* DefaultCursors)
    {
    }
```

```
C++ Builder void __fastcall OLEGiveFeedback(TObject *Sender,long Effect,VARIANT_BOOL *
    DefaultCursors)
    {
    }
```

```
Delphi procedure OLEGiveFeedback(ASender: TObject; Effect : Integer;var DefaultCursors
    : WordBool);
begin
end;
```

```
Delphi 8 (.NET only) procedure OLEGiveFeedback(sender: System.Object; e:
    AxEXSURFACELib._ISurfaceEvents_OLEGiveFeedbackEvent);
begin
end;
```

```
Powe... begin event OLEGiveFeedback(long Effect,boolean DefaultCursors)
end event OLEGiveFeedback
```

VB.NET

```
Private Sub OLEGiveFeedback(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_OLEGiveFeedbackEvent) Handles OLEGiveFeedback  
End Sub
```

VB6

```
Private Sub OLEGiveFeedback(ByVal Effect As Long,DefaultCursors As Boolean)  
End Sub
```

VBA

```
Private Sub OLEGiveFeedback(ByVal Effect As Long,DefaultCursors As Boolean)  
End Sub
```

VFP

```
LPARAMETERS Effect,DefaultCursors
```

Xbas...

```
PROCEDURE OnOLEGiveFeedback(oSurface,Effect,DefaultCursors)  
RETURN
```

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

Java...

```
<SCRIPT EVENT="OLEGiveFeedback(Effect,DefaultCursors)"  
LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function OLEGiveFeedback(Effect,DefaultCursors)  
End Function  
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComOLEGiveFeedback Integer lEffect Boolean lDefaultCursors  
Forward Send OnComOLEGiveFeedback lEffect lDefaultCursors  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_OLEGiveFeedback(Effect,DefaultCursors) CLASS MainDialog  
RETURN NIL
```

X++

```
// OLEGiveFeedback event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

**XBasic**

```
function OLEGiveFeedback as v (Effect as N,DefaultCursors as L)
end function
```

**dBASE**

```
function nativeObject_OLEGiveFeedback(Effect,DefaultCursors)
return
```

# event OLESetData (Data as ExDataObject, Format as Integer)

Occurs on a drag source when a drop target calls the GetData method and there is no data in a specified format in the OLE drag-and-drop DataObject.

Type	Description
Data as <a href="#">ExDataObject</a>	An ExDataObject object in which to place the requested data. The component calls the SetData method to load the requested format.
Format as Integer	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the ExDataObject object.

The OLESetData is not implemented

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

C#

// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.

VB

// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.

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

C#

private void OLESetData(object sender, AxEXSURFACELib.\_ISurfaceEvents\_OLESetDataEvent e)  
{  
}

C++

void OnOLESetData(LPDISPATCH Data,short Format)  
{  
}

C++ Builder

void \_\_fastcall OLESetData(TObject \*Sender,Exsurfacelib\_tlb::IExDataObject \*Data,short Format)  
{  
}

Delphi

```
procedure OLESetData(ASender: TObject; Data : IExDataObject;Format : Smallint);  
begin  
end;
```

Delphi 8  
(.NET  
only)

```
procedure OLESetData(sender: System.Object; e:  
AxEXSURFACELib._ISurfaceEvents_OLESetDataEvent);  
begin  
end;
```

Power...

```
begin event OLESetData(oleobject Data,integer Format)  
end event OLESetData
```

VB.NET

```
Private Sub OLESetData(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_OLESetDataEvent) Handles OLESetData  
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXSURFACELibCtl.IExDataObject,ByVal  
Format As Integer)  
End Sub
```

VBA

```
Private Sub OLESetData(ByVal Data As Object,ByVal Format As Integer)  
End Sub
```

VFP

```
LPARAMETERS Data,Format
```

Xbas...

```
PROCEDURE OnOLESetData(oSurface,Data,Format)  
RETURN
```

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

Java...

```
<SCRIPT EVENT="OLESetData(Data,Format)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function OLESetData(Data,Format)  
End Function
```

```
</SCRIPT>
```

Visual  
Data...

```
Procedure OnComOLESetData Variant IIData Short IIDFormat  
    Forward Send OnComOLESetData IIData IIDFormat  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_OLESetData(Data,Format) CLASS MainDialog  
RETURN NIL
```

X++

```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

XBasic

```
function OLESetData as v (Data as OLE::Exontrol.Surface.1::IExDataObject,Format  
as N)  
end function
```

dBASE

```
function nativeObject_OLESetData(Data,Format)  
return
```



## event **OLEStartDrag** (Data as **ExDataObject**, AllowedEffects as **Long**)

Occurs when the **OLEDrag** method is called.

Type	Description
Data as <a href="#">ExDataObject</a>	An <b>ExDataObject</b> object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the <b>ExDataObject</b> , it is provided when the control calls the <b>GetData</b> method. The programmer should provide the values for this parameter in this event. The <b>SetData</b> and <b>Clear</b> methods cannot be used here.
AllowedEffects as <b>Long</b>	A <b>long</b> containing the effects that the source component supports. The possible values are listed in <b>Settings</b> . The programmer should provide the values for this parameter in this event

*In the /NET Assembly, you have to use the **DragEnter** event as explained here:*

- <https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop>

The settings for **AllowEffects** are:

- **exOLEDropEffectNone** (0), Drop target cannot accept the data, or the drop operation was cancelled
- **exOLEDropEffectCopy** (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- **exOLEDropEffectMove** (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

The source component should logically Or together the supported values and places the result in the **AllowedEffects** parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be). You may wish to defer putting data into the **ExDataObject** object until the target component requests it. This allows the source component to save time. If the user does not load any formats into the **ExDataObject**, then the drag/drop operation is canceled. Use [exCFFiles](#) and [Files](#) property to add files to the drag and drop data object.

The idea of drag and drop in **exSurface** control is the same as in other controls. To start accepting drag and drop sources the **exSurface** control should have the [OLEDropMode](#) to **exOLEDropManual**. Once that is set, the **exSurface** starts accepting any drag and drop sources.

The first step is if you want to be able to drag items from your **exSurface** control to other

controls the idea is to handle the `OLE_StartDrag` event. The event passes an object `ExDataObject` (Data) as argument. The Data and AllowedEffects can be changed only in the `OLEStartDrag` event. The `OLE_StartDrag` event is fired when user is about to drag items from the control. **The AllowedEffect parameter and [SetData](#) property must be set to continue drag and drop operation, as in the following samples:**

Syntax for `OLEStartDrag` event, **/NET** version, on:

```
C# // OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,
    DragDrop ... events.
```

```
VB // OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,
    DragDrop ... events.
```

Syntax for `OLEStartDrag` event, **/COM** version, on:

```
C# private void OLEStartDrag(object sender,
    AxEXSURFACELib._ISurfaceEvents_OLEStartDragEvent e)
    {
    }
```

```
C++ void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
    {
    }
```

```
C++ Builder void __fastcall OLEStartDrag(TObject *Sender,Exsurfacelib_tlb::IExDataObject
    *Data,long * AllowedEffects)
    {
    }
```

```
Delphi procedure OLEStartDrag(ASender: TObject; Data : IExDataObject;var
    AllowedEffects : Integer);
begin
end;
```

```
Delphi 8 (.NET only) procedure OLEStartDrag(sender: System.Object; e:
    AxEXSURFACELib._ISurfaceEvents_OLEStartDragEvent);
begin
end;
```

**Powe...** begin event OLEStartDrag(oleobject Data,long AllowedEffects)  
end event OLEStartDrag

**VB.NET** Private Sub OLEStartDrag(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_OLEStartDragEvent) Handles OLEStartDrag  
End Sub

**VB6** Private Sub OLEStartDrag(ByVal Data As  
EXSURFACELibCtl.IExDataObject,AllowedEffects As Long)  
End Sub

**VBA** Private Sub OLEStartDrag(ByVal Data As Object,AllowedEffects As Long)  
End Sub

**VFP** LPARAMETERS Data,AllowedEffects

**Xbas...** PROCEDURE OnOLEStartDrag(oSurface,Data,AllowedEffects)  
RETURN

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

**Java...** <SCRIPT EVENT="OLEStartDrag(Data,AllowedEffects)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function OLEStartDrag(Data,AllowedEffects)  
End Function  
</SCRIPT>

**Visual  
Data...** Procedure OnComOLEStartDrag Variant IData Integer IIAAllowedEffects  
Forward Send OnComOLEStartDrag IData IIAAllowedEffects  
End\_Procedure

**Visual  
Objects** METHOD OCX\_OLEStartDrag(Data,AllowedEffects) CLASS MainDialog  
RETURN NIL

**X++** // OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,

DragDrop ... events.

XBasic

```
function OLEStartDrag as v (Data as  
OLE::Exontrol.Surface.1::IExDataObject, AllowedEffects as N)  
end function
```

dBASE

```
function nativeObject_OLEStartDrag(Data, AllowedEffects)  
return
```

The following VB sample drags data from a control to another, by registering a new clipboard format:

```
Private Sub Surface1_OLEStartDrag(Index As Integer, ByVal Data As  
EXSURFACELibCtl.IExDataObject, AllowedEffects As Long)
```

```
' We are going to add two clipboard formats: text and "EXSURFACE" clipboard format.  
' We need to use RegisterClipboardFormat API function in order to register our  
' clipboard format. One clipboard format is enough, but the sample shows  
' how to filter in OLEDragDrop event the other clipboard formats
```

```
' Builds a string that contains each cell's caption on a new line
```

```
Dim n As Long
```

```
Dim s As String
```

```
With Surface1(Index)
```

```
    s = Index & vbCrLf ' Saves the source
```

```
    For n = 0 To .Columns.Count - 1
```

```
        s = s & .Items.CellCaption(.Items.SelectedItem(0), n) & vbCrLf
```

```
    Next
```

```
End With
```

```
AllowedEffects = 0
```

```
' Checks whether the selected item has a parent
```

```
If (Surface1(Index).Items.ItemParent(Surface1(Index).Items.SelectedItem(0)) <> 0) Then
```

```
    AllowedEffects = 1
```

```
End If
```

```
' Sets the text clipboard format
```

```
Data.SetData s, exCFTText
```

- ' Builds an array of bytes, and copy there all characters in the s string.
- ' Passes the array to the SetData method.

```
ReDim v(Len(s)) As Byte
For n = 0 To Len(s) - 1
    v(n) = Asc(Mid(s, n + 1, 1))
Next
Data.SetData v, RegisterClipboardFormat("EXSURFACE")
```

End Sub

The code fills data for two types of clipboard formats: text ( CF\_TEXT ) and "EXSURFACE" registered clipboard format. The registered clipboard format must be an array of bytes. As you can see we have used the RegisterClipboardFormat API function, and it should be declared like:

```
Private Declare Function RegisterClipboardFormat Lib "user32" Alias
"RegisterClipboardFormatA" (ByVal lpString As String) As Integer
```

The second step is accepting OLE drag and drop source objects. That means, if you would like to let your control accept drag and drop objects, you have to handle the [OLEDragDrop](#) event. It gets as argument an object Data that stores the drag and drop information. The next sample shows how handle the OLEDragDrop event:

```
Private Sub Surface1_OLEDragDrop(Index As Integer, ByVal Data As
EXSURFACELibCtl.IExDataObject, Effect As Long, ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
    ' Checks whether the clipboard format is our. Since we have registered the clipboard in
the
    ' OLEStartData format we now its format, so we can handle this type of clip formats.
    If (Data.GetFormat(RegisterClipboardFormat("EXSURFACE"))) Then
        ' Builds the saved string from the array passed
        Dim s As String
        Dim v() As Byte
        Dim n As Integer
        v = Data.GetData(RegisterClipboardFormat("EXSURFACE"))
        For n = LBound(v) To UBound(v)
            s = s + Chr(v(n))
        Next
```

Debug.Print s

'Adds a new item to the control, and sets the cells captions like we saved, line by line

Surface1(Index).Visible = False

With Surface1(Index)

.BeginUpdate

Dim i As HITEM

Dim item As String

Dim nCur As Long

i = .Items.AddItem()

nCur = InStr(1, s, vbCrLf) + Len(vbCrLf) ' Jumps the source

For n = 0 To .Columns.Count - 1

Dim nnCur As Long

nnCur = InStr(nCur, s, vbCrLf)

.Items.CellCaption(i, n) = Mid(s, nCur, nnCur - nCur)

nCur = nnCur + Len(vbCrLf)

Next

.Items.CellImage(i, "EmployeeID") = Int(.Items.CellCaption(i, "EmployeeID"))

.Items.SetParent i, h(Index, Int(.Items.CellCaption(i, "EmployeeID")) - 1)

.Items.EnsureVisibleItem i

.EndUpdate

End With

Surface1(Index).Visible = True

End If

End Sub

The following VC sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
#import <exsurface.dll> rename( "GetItems", "exGetItems" )
```

```
#include "Items.h"
```

```
#include "Columns.h"
```

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
```

```
{
```

```
    if ( pv )
```

```
    {
```

```

    if ( pv->vt == VT_ERROR )
        return szDefault;

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

void OnOLEStartDragSurface1(LPDISPATCH Data, long FAR* AllowedEffects)
{
    CItems items = m_surface.GetItems();
    long nCount = items.GetSelectCount(), nColumnCount =
m_surface.GetColumns().GetCount();
    if ( nCount > 0 )
    {
        *AllowedEffects = /*exOLEDropEffectCopy */ 1;
        EXSURFACELib::IExDataObjectPtr spData( Data );
        if ( spData != NULL )
        {
            CString strData;
            for ( long i = 0; i < nCount; i++ )
            {
                COleVariant vtlItem( items.GetSelectedItem( i ) );
                for ( long j = 0; j < nColumnCount; j++ )
                    strData += V2S( &items.GetCellCaption( vtlItem, COleVariant( j ) ) ) + "\t";
            }
            strData += "\r\n";
            spData->SetData( COleVariant( strData ), COleVariant(
(long)EXSURFACELib::exCFText ) );
        }
    }
}

```

The sample saves data as CF\_TEXT format ( EXSURFACELib::exCFText ). The data is a text, where each item is separated by "\r\n" ( new line ), and each cell is separated by "\t" (

TAB character ). Of course, data can be saved as you want. The sample only gives an idea of what and how it could be done. The sample uses the `#import` statement to import the control's type library, including definitions for [ExDataObject](#) and [ExDataObjectFiles](#) that are required to fill data to be dragged. If your `exsurface.dll` file is located in another place than your system folder, the path to the `exsurface.dll` file needs to be specified, else compiler errors occur.

The following VB.NET sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
Private Sub AxSurface1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_OLEStartDragEvent) Handles AxSurface1.OLEStartDrag
    With AxSurface1.Items
        If (.SelectCount > 0) Then
            e.allowedEffects = 1 'exOLEDropEffectCopy
            Dim i As Integer, j As Integer, strData As String, nColumnCount As Long =
AxSurface1.Columns.Count
            For i = 0 To .SelectCount - 1
                For j = 0 To nColumnCount - 1
                    strData = strData + .CellCaption(.SelectedItem(i), j) + Chr(Keys.Tab)
                Next
            Next
            strData = strData + vbCrLf
            e.data.SetData(strData, EXSURFACELib.exClipboardFormatEnum.exCFText)
        End If
    End With
End Sub
```

The following C# sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
private void axSurface1_OLEStartDrag(object sender,
AxEXSURFACELib._ISurfaceEvents_OLEStartDragEvent e)
{
    int nCount = axSurface1.Items.SelectCount;
    if ( nCount > 0 )
    {
        int nColumnCount = axSurface1.Columns.Count;
        e.allowedEffects = /*exOLEDropEffectCopy*/ 1;
```



```

string strData = "";
for ( int i =0 ; i < nCount; i++ )
{
    for ( int j = 0; j < nColumnCount; j++ )
    {
        object strCell =
axSurface1.Items.get_CellCaption(axSurface1.Items.get_SelectedItem(i), j);
        strData += ( strCell != null ? strCell.ToString() : "" ) + "\t";
    }
    strData += "\r\n";
}
e.data.SetData( strData, EXSURFACELib.exClipboardFormatEnum.exCFText );
}
}

```

The following VFP sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```

*** ActiveX Control Event ***
LPARAMETERS data, allowedeffects

local sData, nColumnCount, i, j
with thisform.Surface1.Items
    if ( .SelectCount() > 0 )
        allowedeffects = 1 && exOLEDropEffectCopy
        sData = ""
        nColumnCount = thisform.Surface1.Columns.Count
        for i = 0 to .SelectCount - 1
            for j = 0 to nColumnCount
                sData = sData + .CellCaption( .SelectedItem(i), j ) + chr(9)
            next
            sData = sData + chr(10)+ chr(13)
        next
        data.SetData( sData, 1 ) && exCFText
    endif
endwith

```

# event ParentChangeEvent (Element as Element)

The element is expanded or collapsed.

Type	Description
Element as <a href="#">Element</a>	An Element object whose parent is changed.

The ParentChangeEvent event occurs when the element's parent is changed. Use the [AllowInsertObject](#) property to specify whether the user can change the element's parent at runtime by dragging the element over the other. The [Parent](#) property indicates the element's parent. The [AllowInsertChild](#) property of the Element object specifies whether the element supports adding child elements at runtime. The [AllowChangeParent](#) property of the Element object specifies whether the element can change its parent at runtime.

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

```
C# private void ParentChangeEvent(object sender,exontrol.EXSURFACELib.Element
Element)
{
}

VB Private Sub ParentChangeEvent(ByVal sender As System.Object,ByVal Element
As exontrol.EXSURFACELib.Element) Handles ParentChangeEvent
End Sub
```

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

```
C# private void ParentChangeEvent(object sender,
AxEXSURFACELib._ISurfaceEvents_ParentChangeEvent e)
{
}

C++ void OnParentChangeEvent(LPDISPATCH Element)
{
}

C++ Builder void __fastcall ParentChangeEvent(TObject *Sender,Exsurfacelib_tlb::IElement
*Element)
{
}
```

**Delphi** procedure ParentChangeElement(ASender: TObject; Element : IElement);  
begin  
end;

**Delphi 8  
(.NET  
only)** procedure ParentChangeElement(sender: System.Object; e:  
AxEXSURFACELib.\_ISurfaceEvents\_ParentChangeElementEvent);  
begin  
end;

**Powe...** begin event ParentChangeElement(oleobject Element)  
end event ParentChangeElement

**VB.NET** Private Sub ParentChangeElement(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_ParentChangeElementEvent) Handles  
ParentChangeElement  
End Sub

**VB6** Private Sub ParentChangeElement(ByVal Element As EXSURFACELibCtl.IElement)  
End Sub

**VBA** Private Sub ParentChangeElement(ByVal Element As Object)  
End Sub

**VFP** LPARAMETERS Element

**Xbas...** PROCEDURE OnParentChangeElement(oSurface,Element)  
RETURN

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

**Java...** <SCRIPT EVENT="ParentChangeElement(Element)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function ParentChangeElement(Element)  
End Function  
</SCRIPT>

Visual  
Data...

```
Procedure OnComParentChangeElement Variant IElement  
    Forward Send OnComParentChangeElement IElement  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_ParentChangeElement(Element) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_ParentChangeElement(COM _Element)  
{  
}
```

XBasic

```
function ParentChangeElement as v (Element as OLE::Exontrol.Surface.1::IElement)  
end function
```

dBASE

```
function nativeObject_ParentChangeElement(Element)  
return
```

# event RClick ()

Occurs once the user right clicks the control.

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

The RClick event notifies when the user right clicks the control. The [ElementFromPoint](#)(-1,-1) property returns the element from the cursor or nothing if no element at the cursor position. The [HitTestFromPoint](#) property returns the element and the hit-test code from the cursor. You can use the [Edit](#) method to edit the element's caption or extra caption. The [HandCursorClick](#) event notifies once the user clicks a part of the element ( which shows a hand cursor when the pointer hovers it ).

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;
```

Power...

```
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(oSurface)  
RETURN
```

Syntax for RClick event, **/COM** 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 RemoveElement (Element as Element)

An element has been removed from the surface.

Type	Description
Element as <a href="#">Element</a>	An Element object that specifies the element being removed from the <a href="#">Elements</a> collection.

The RemoveElement event occurs once the element has been removed from the [Elements](#) collection. Use the RemoveElement event to release any extra data associated with the element. Use the [Remove](#) method to remove a specific element from the Elements collection. Use the [RemoveSelection](#) method to remove the selected elements. Use the [Clear](#) method to remove all elements from the surface. Removing an element removes the incoming and outgoing links as well.

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

C#	<pre>private void RemoveElement(object sender,exontrol.EXSURFACELib.Element Element) { }</pre>
VB	<pre>Private Sub RemoveElement(ByVal sender As System.Object,ByVal Element As exontrol.EXSURFACELib.Element) Handles RemoveElement End Sub</pre>

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

C#	<pre>private void RemoveElement(object sender, AxEXSURFACELib._ISurfaceEvents_RemoveElementEvent e) { }</pre>
C++	<pre>void OnRemoveElement(LPDISPATCH Element) { }</pre>
C++ Builder	<pre>void __fastcall RemoveElement(TObject *Sender,Exsurfacelib_tlb::IElement *Element) { }</pre>



**Delphi** procedure RemoveElement(ASender: TObject; Element : IElement);  
begin  
end;

**Delphi 8  
(.NET  
only)** procedure RemoveElement(sender: System.Object; e:  
AxEXSURFACELib.\_ISurfaceEvents\_RemoveElementEvent);  
begin  
end;

**Powe...** begin event RemoveElement(oleobject Element)  
end event RemoveElement

**VB.NET** Private Sub RemoveElement(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib.\_ISurfaceEvents\_RemoveElementEvent) Handles RemoveElement  
End Sub

**VB6** Private Sub RemoveElement(ByVal Element As EXSURFACELibCtl.IElement)  
End Sub

**VBA** Private Sub RemoveElement(ByVal Element As Object)  
End Sub

**VFP** LPARAMETERS Element

**Xbas...** PROCEDURE OnRemoveElement(oSurface,Element)  
RETURN

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

**Java...** <SCRIPT EVENT="RemoveElement(Element)" LANGUAGE="JScript">  
</SCRIPT>

**VBSc...** <SCRIPT LANGUAGE="VBScript">  
Function RemoveElement(Element)  
End Function  
</SCRIPT>

Visual  
Data...

```
Procedure OnComRemoveElement Variant IElement  
    Forward Send OnComRemoveElement IElement  
End_Procedure
```

Visual  
Objects

```
METHOD OCX_RemoveElement(Element) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_RemoveElement(COM _Element)  
{  
}
```

XBasic

```
function RemoveElement as v (Element as OLE::Exontrol.Surface.1::IElement)  
end function
```

dBASE

```
function nativeObject_RemoveElement(Element)  
return
```

# event RemoveLink (Link as Link)

The link is removed from the links collection.

Type	Description
Link as <a href="#">Link</a>	A Link object that specifies the link to be removed.

The RemoveLink event notifies your application once a link has been removed from the [Links](#) collection. The [Remove](#) method removes a specific link from the Links collection. Use the [Clear](#) method to remove all links from the surface. The [IncomingLinks](#) property returns a safe array of incoming links ( links that ends on the element ). The [OutgoingLinks](#) property returns a safe array of outgoing links ( links that starts from the element ).

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

C#	<pre>private void RemoveLink(object sender,exontrol.EXSURFACELib.Link Link) { }</pre>
VB	<pre>Private Sub RemoveLink(ByVal sender As System.Object,ByVal Link As exontrol.EXSURFACELib.Link) Handles RemoveLink End Sub</pre>

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

C#	<pre>private void RemoveLink(object sender, AxEXSURFACELib._ISurfaceEvents_RemoveLinkEvent e) { }</pre>
C++	<pre>void OnRemoveLink(LPDISPATCH Link) { }</pre>
C++ Builder	<pre>void __fastcall RemoveLink(TObject *Sender,Exsurfacelib_tlb::ILink *Link) { }</pre>
Delphi	<pre>procedure RemoveLink(ASender: TObject; Link : ILink); begin end;</pre>

Delphi 8  
(.NET  
only)

```
procedure RemoveLink(sender: System.Object; e:  
AxEXSURFACELib._ISurfaceEvents_RemoveLinkEvent);  
begin  
end;
```

Powe...

```
begin event RemoveLink(oleobject Link)  
end event RemoveLink
```

VB.NET

```
Private Sub RemoveLink(ByVal sender As System.Object, ByVal e As  
AxEXSURFACELib._ISurfaceEvents_RemoveLinkEvent) Handles RemoveLink  
End Sub
```

VB6

```
Private Sub RemoveLink(ByVal Link As EXSURFACELibCtl.ILink)  
End Sub
```

VBA

```
Private Sub RemoveLink(ByVal Link As Object)  
End Sub
```

VFP

```
LPARAMETERS Link
```

Xbas...

```
PROCEDURE OnRemoveLink(oSurface,Link)  
RETURN
```

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

Java...

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

VBSc...

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

Visual  
Data...

```
Procedure OnComRemoveLink Variant IILink  
    Forward Send OnComRemoveLink IILink  
End_Procedure
```

Visual  
Objects

METHOD OCX\_RemoveLink(Link) CLASS MainDialog  
RETURN NIL

X++

```
void onEvent_RemoveLink(COM _Link)
{
}
```

XBasic

```
function RemoveLink as v (Link as OLE::Exontrol.Surface.1::ILink)
end function
```

dBASE

```
function nativeObject_RemoveLink(Link)
return
```

# event SelectionChanged ()

Notifies your application that the control's selection has been changed.

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

The SelectionChanged event occurs once a new element is selected or unselected. The [Selected](#) property of the Element object indicates whether the element is selected or unselected. The [Selectable](#) property of the Element object indicates whether the element is selectable or un-selectable.

The [SelectObjectColor](#) / [SelectObjectTextColor](#) property specifies the colors to show the selected elements ( while the control has the focus ). The [SelectObjectColorInactive](#) / [SelectObjectTextColorInactive](#) property specifies the color to show the selected elements ( while the control is not focused ). The [SelectObjectStyle](#) property specifies the style to show the selected elements ( like changing the element's background/foreground colors, showing a border around the selected elements, and so on ). Use the [Background](#)(exSelectObjectRectColor) property to specify the color to show the rectangle that highlights the elements that intersect the dragging rectangle.

The [SingleSel](#) property specifies whether the surface allows selecting one or multiple elements. The [SelCount](#) property counts the number of selected elements. The [SelElement](#) property returns the selected element based on its index in the selected elements collection. The [Selection](#) property sets or gets a safe array of selected elements. The [AllowSelectObject](#) property indicates the keys combination to allow user selecting new elements. The [AllowSelectObjectRect](#) property specifies the keys combination so the user can select the elements from the dragging rectangle. The [AllowSelectNothing](#) property indicates whether the selection is cleared once the user clicks any empty area on the surface. The [SelectAll](#) method selects all elements in the chart. Use the [UnselectAll](#) method to unselect all elements on the surface.

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

```
C# private void SelectionChanged(object sender)
{
}
```

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

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

**C#**

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

**C++**

```
void OnSelectionChanged()
{
}
```

**C++  
Builder**

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

**Delphi**

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

**Delphi 8  
(.NET  
only)**

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

**Powe...**

```
begin event SelectionChanged()
end event SelectionChanged
```

**VB.NET**

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

**VB6**

```
Private Sub SelectionChanged()  
End Sub
```

**VBA**

```
Private Sub SelectionChanged()  
End Sub
```

**VFP**

```
LPARAMETERS nop
```

**Xbas...**

```
PROCEDURE OnSelectionChanged(oSurface)
```

## RETURN

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

Java... `<SCRIPT EVENT="SelectionChanged()" LANGUAGE="JScript">  
</SCRIPT>`

VBSc... `<SCRIPT LANGUAGE="VBScript">  
Function SelectionChanged()  
End Function  
</SCRIPT>`

Visual  
Data... `Procedure OnComSelectionChanged  
    Forward Send OnComSelectionChanged  
End_Procedure`

Visual  
Objects `METHOD OCX_SelectionChanged() CLASS MainDialog  
RETURN NIL`

X++ `void onEvent_SelectionChanged()  
{  
}  
}`

XBasic `function SelectionChanged as v ()  
end function`

dBASE `function nativeObject_SelectionChanged()  
return`

The following VB sample enumerates the incoming elements ( of selected elements ):

```
Private Sub Surface1_SelectionChanged()  
    With Surface1  
        Dim s As Variant  
        For Each s In .Selection  
            Debug.Print "Incomming Elements of " & s.ID & "are: "  
        With s
```



```
        For Each i In .IncomingLinks
            Debug.Print i.ElementFrom.ID
        Next
    End With
Next
End With
End Sub
```

The following VB sample enumerates the outgoing elements ( of selected elements ):

```
Private Sub Surface1_SelectionChanged()
    With Surface1
        Dim s As Variant
        For Each s In .Selection
            Debug.Print "Outgoing Elements of " & s.ID & "are: "
            With s
                For Each i In .OutgoingLinks
                    Debug.Print i.ElementTo.ID
                Next
            End With
        Next
    End With
Next
End With
End Sub
```

# event **ToolBarAnchorClick** (AnchorID as String, Options as String)

Occurs when an anchor element is clicked, on the control's toolbar.

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 **ToolBarAnchorClick** event to notify that the user clicks an anchor element ( being displayed on the control's toolbar ). 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 **ToolBarAnchorClick** event is fired only if prior clicking the control it shows the hand cursor. The control fires the **ToolBarAnchorClick** event when the user clicks an hyperlink element. The [ToolBarCaption](#) property specifies the caption of the button. Use the [ToolBarRefresh](#) method to refresh the control's toolbar. The [ToolBarToolTip](#) property specifies the button's tooltip. The control fires the [ToolBarClick](#) event when the user clicks a button in the surface's toolbar.

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

```
C# private void ToolBarAnchorClick(object sender,string AnchorID,string Options)
{
}
```

```
VB Private Sub ToolBarAnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles ToolBarAnchorClick
End Sub
```

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

```
C# private void ToolBarAnchorClick(object sender,
AxEXSURFACELib._ISurfaceEvents_ToolBarAnchorClickEvent e)
{
}
```

```
C++ void OnToolBarAnchorClick(LPCTSTR AnchorID,LPCTSTR Options)
{
}
```

C++  
Builder

```
void __fastcall ToolBarAnchorClick(TObject *Sender,BSTR AnchorID,BSTR Options)
{
}
```

Delphi

```
procedure ToolBarAnchorClick(ASender: TObject; AnchorID : WideString;Options :
WidesString);
begin
end;
```

Delphi 8  
(.NET  
only)

```
procedure ToolBarAnchorClick(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_ToolBarAnchorClickEvent);
begin
end;
```

Powe...

```
begin event ToolBarAnchorClick(string AnchorID,string Options)
end event ToolBarAnchorClick
```

VB.NET

```
Private Sub ToolBarAnchorClick(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_ToolBarAnchorClickEvent) Handles
ToolBarAnchorClick
End Sub
```

VB6

```
Private Sub ToolBarAnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

VBA

```
Private Sub ToolBarAnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

VFP

```
LPARAMETERS AnchorID,Options
```

Xbas...

```
PROCEDURE OnToolBarAnchorClick(oSurface,AnchorID,Options)
RETURN
```

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

Java...

```
<SCRIPT EVENT="ToolBarAnchorClick(AnchorID,Options)" LANGUAGE="JScript">
</SCRIPT>
```

**VBS...**

```
<SCRIPT LANGUAGE="VBScript">  
Function ToolBarAnchorClick(AnchorID,Options)  
End Function  
</SCRIPT>
```

**Visual  
Data...**

```
Procedure OnComToolBarAnchorClick String IIAnchorID String IIOptions  
    Forward Send OnComToolBarAnchorClick IIAnchorID IIOptions  
End_Procedure
```

**Visual  
Objects**

```
METHOD OCX_ToolBarAnchorClick(AnchorID,Options) CLASS MainDialog  
RETURN NIL
```

**X++**

```
void onEvent_ToolBarAnchorClick(str _AnchorID,str _Options)  
{  
}  
}
```

**XBasic**

```
function ToolBarAnchorClick as v (AnchorID as C,Options as C)  
end function
```

**dBASE**

```
function nativeObject_ToolBarAnchorClick(AnchorID,Options)  
return
```

# event ToolBarClick (ID as Long, SelectedID as Long)

Occurs when the user clicks a button in the toolbar.

Type	Description
ID as Long	A Long expression that specifies the identifier of the button/selector being clicked.
SelectedID as Long	A long expression that specifies the identifier being selected. ( the identifier being specified by the second part of the <a href="#">ToolBarCaption</a> property [separated by # character ] ). For instance, if the ToolBarCaption property is "Letter#1234" the button displays the "Letter" label, the SelectedID parameter is 1234 if the user clicks the button or selects the item in a drop down field.

The control fires the ToolBarClick event when the user clicks a button or selects a value in the control's toolbar. The [ToolBarCaption](#) property specifies the caption of the button. The [ToolBarToolTip](#) property specifies the button's tooltip. Use the [ToolBarRefresh](#) method to refresh the control's toolbar. Use the [ToolBarFormat](#) property to add new buttons, to display icons, pictures, or any other HTML caption.

The following screen shot shows the control's default toolbar:



For instance, clicking the Home button, generates the ToolBarClick(100) event. Instead selecting a new value from the zoom field ( drop-down field ), generates ToolBarClick(101, zoom) where the zoom is the zoom-factor being chosen.

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

C#

```
private void ToolBarClick(object sender,int ID,int SelectedID)
{
}
```

VB

```
Private Sub ToolBarClick(ByVal sender As System.Object,ByVal ID As Integer,ByVal
SelectedID As Integer) Handles ToolBarClick
End Sub
```

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

C#

```
private void ToolBarClick(object sender,
```

```
AxEXSURFACELib._ISurfaceEvents_ToolBarClickEvent e)
{
}
```

C++

```
void OnToolBarClick(long ID,long SelectedID)
{
}
```

C++  
Builder

```
void __fastcall ToolBarClick(TObject *Sender,long ID,long SelectedID)
{
}
```

Delphi

```
procedure ToolBarClick(ASender: TObject; ID : Integer;SelectedID : Integer);
begin
end;
```

Delphi 8  
(.NET  
only)

```
procedure ToolBarClick(sender: System.Object; e:
AxEXSURFACELib._ISurfaceEvents_ToolBarClickEvent);
begin
end;
```

Power...

```
begin event ToolBarClick(long ID,long SelectedID)
end event ToolBarClick
```

VB.NET

```
Private Sub ToolBarClick(ByVal sender As System.Object, ByVal e As
AxEXSURFACELib._ISurfaceEvents_ToolBarClickEvent) Handles ToolBarClick
End Sub
```

VB6

```
Private Sub ToolBarClick(ByVal ID As Long,ByVal SelectedID As Long)
End Sub
```

VBA

```
Private Sub ToolBarClick(ByVal ID As Long,ByVal SelectedID As Long)
End Sub
```

VFP

```
LPARAMETERS ID,SelectedID
```

Xbas...

```
PROCEDURE OnToolBarClick(oSurface,ID,SelectedID)
```

## RETURN

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

Java... <SCRIPT EVENT="ToolBarClick(ID,SelectedID)" LANGUAGE="JScript">  
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">  
Function ToolBarClick(ID,SelectedID)  
End Function  
</SCRIPT>

Visual  
Data... Procedure OnComToolBarClick Integer IID Integer ISelectedID  
Forward Send OnComToolBarClick IID ISelectedID  
End\_Procedure

Visual  
Objects METHOD OCX\_ToolBarClick(ID,SelectedID) CLASS MainDialog  
RETURN NIL

X++ void onEvent\_ToolBarClick(int \_ID,int \_SelectedID)  
{  
}

XBasic function ToolBarClick as v (ID as N,SelectedID as N)  
end function

dBASE function nativeObject\_ToolBarClick(ID,SelectedID)  
return

# property Elements.Item (ID as Variant) as Element

Returns a specific Element of the Elements collection, giving its identifier.

Type	Description
ID as Variant	A Long, String or Numeric expression that defines the element's unique identifier.
<a href="#">Element</a>	An Element object being retrieved.

The Item property accesses the element giving its identifier. The [Count](#) property specifies the number of elements in the Elements collection. The [ID](#) property specifies the element's unique identifier.

The following VB sample enumerates the elements on the surface:

```
Dim e As Variant
For Each e In Surface1.Elements
    Debug.Print e.ID
Next
```



# Expressions

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

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

Usage examples:

- `100 + 200`, adds two numbers and returns `300`
- `"100" + 200`, concatenates the strings, and returns `"100200"`
- `currency(1000)` displays the value in currency format based on the current regional setting, such as `"$1,000.00"` for US format.
- `1000 format ``` gets `1,000.00` for English format, while `1.000,00` is displayed for German format
- `1000 format `2|.|3|,`` always gets `1,000.00` no matter of settings in the control panel.
- `date(value) format `MMM d, yyyy``, returns the date such as `Sep 2, 2023`, for English format
- `upper("string")` converts the giving string in uppercase letters, such as `"STRING"`
- `date(dateS('3/1/' + year(9:=#1/1/2018#)) + ((1:=(((255 - 11 * (year(=9) mod 19)) - 21) mod 30) + 21) + (=:1 > 48 ? -1 : 0) + 6 - ((year(=9) + int(year(=9) / 4)) + =:1 + (=:1 > 48 ? -1 : 0) + 1) mod 7))` returns the date the Easter Sunday will fall, for year 2018. In this case the expression returns `#4/1/2018#`. If `#1/1/2018#` is replaced with `#1/1/2019#`, the expression returns `#4/21/2019#`.

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

*The constants can be represented as:*

- numbers in **decimal** format ( where dot character specifies the decimal separator ). For instance: `-1`, `100`, `20.45`, `.99` and so on
- numbers in **hexa-decimal** format ( preceded by `0x` or `0X` sequence ), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and

programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, `0xFF`, `0x00FF00`, and so so.

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

*The predefined constants are:*

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

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

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

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

**variable := expression**

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

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

**=: variable**

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

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

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

*expression ? true\_part : false\_part*

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

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

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

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

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

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

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

, where the `c1`, `c2`, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the `value in (11,22,33,44,13)` is equivalent with `(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)`. The `in` operator is not a time consuming as the equivalent `or` version is, so when you have large number of constant elements it is recommended using the

*in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

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

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

, where the *c1*, *c2*, ... are constant elements, and the *default* is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : ( %0 = c 2 ? c 2 : ( ... ? . : default) )". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker 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 *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. The type operator may return any of the following: 0 - empty ( not initialized ), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, **7 - date**, **8 - string**, 9 - object, 10 - error, **11 - boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bits. For instance `type(%1) = 8` specifies the cells ( on the column with the index 1 ) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the `str(-12.54)` returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the `dbl("12.54")` returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the `date(``)` gets the current date ( no time included ), the `date(`now`)` gets the current date-time, while the `date("01/01/2001")` returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the `dateS("01/01/2001 14:00:00")` returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, `hex(`FF`)` returns 255, while the `hex(255)` or `hex(0xFF)` returns the `FF` string. The `hex(hex(`FFFFFFFF`))` always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

*The bitwise operators for numbers are:*

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, `0x01001000 bitand 0x10111000` returns 0x00001000.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, `0x01001000 bitor 0x10111000` returns 0x11111000.
- a **bitxor** b (binary operator) computes the XOR ( exclusive-OR ) operation on bits of a and b, and returns the unsigned value. For instance, `0x01110010 bitxor 0x10101010` returns 0x11011000.



- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, `128 bitshift 1` returns 64 ( dividing by 2 ) or `128 bitshift (-1)` returns 256 ( multiplying by 2 )
- **bitnot** ( unary operator ) flips every bit of x, and returns the unsigned value. For instance, `bitnot(0x00FF0000)` returns 0xFF00FFFF.

*The operators for numbers are:*

- **int** (unary operator) retrieves the integer part of the number. For instance, the `int(12.54)` returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the `round(12.54)` returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the `floor(12.54)` returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the `abs(-12.54)` returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the `sin(3.14)` returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the `cos(3.14)` returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the `2*asin(1)` returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the `2*acos(0)` returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the `sqrt(81)` returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, `currency(value)` displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format.
- value **format** 'flags' (binary operator) formats a numeric value with specified flags. The format method formats numeric or date expressions (depends on the type of the value, explained at operators for dates). 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 the settings in your control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

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

- *NumDigits* - specifies the number of fractional digits, If the flag is missing, the

field "No. of digits after decimal" from "Regional and Language Options" is using.

- *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
- *Grouping* - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
  - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
  - 1 - Negative sign, number; for example, -1.1
  - 2 - Negative sign, space, number; for example, - 1.1
  - 3 - Number, negative sign; for example, 1.1-
  - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

*The operators for strings are:*

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"



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

The operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the `time(#1/1/2001 13:00#)` returns "1:00:00 PM"
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the `timeF(#1/1/2001 13:00#)` returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the `shortdate(#1/1/2001 13:00#)` returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the `shortdateF(#1/1/2001 13:00#)` returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the `dateF(#01/01/2001 14:00:00#)` returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the `longdate(#1/1/2001 13:00#)` returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the `year(#12/31/1971 13:14:15#)` returns 1971
- **month** (unary operator) retrieves the month of the date ( 1, 2,...,12 ). For instance, the `month(#12/31/1971 13:14:15#)` returns 12.
- **day** (unary operator) retrieves the day of the date ( 1, 2,...,31 ). For instance, the `day(#12/31/1971 13:14:15#)` returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st ( 0, 1,...,365 ). For instance, the `yearday(#12/31/1971 13:14:15#)` returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday ( 0 - Sunday, 1 - Monday,..., 6 - Saturday ). For instance, the `weekday(#12/31/1971 13:14:15#)` returns 5.
- **hour** (unary operator) retrieves the hour of the date ( 0, 1, ..., 23 ). For instance, the `hour(#12/31/1971 13:14:15#)` returns 13
- **min** (unary operator) retrieves the minute of the date ( 0, 1, ..., 59 ). For instance, the `min(#12/31/1971 13:14:15#)` returns 14
- **sec** (unary operator) retrieves the second of the date ( 0, 1, ..., 59 ). For instance, the `sec(#12/31/1971 13:14:15#)` returns 15
- value **format** 'flags' (binary operator) formats a date expression with specified flags. The format method formats numeric (depends on the type of the value, explained at operators for numbers) or date expressions. If not supported, the value is formatted as a number (the date format is supported by newer version only). The flags specifies the format picture string that is used to form the date. Possible values for the format picture string are defined below. For instance, the `date(value) format 'MMM d, yyyy'`

returns "Sep 2, 2023"

The following table defines the format types used to represent days:

- d, day of the month as digits without leading zeros for single-digit days (8)
- dd, day of the month as digits with leading zeros for single-digit days (08)
- ddd, abbreviated day of the week as specified by the current locale ("Mon" in English)
- dddd, day of the week as specified by the current locale ("Monday" in English)

The following table defines the format types used to represent months:

- M, month as digits without leading zeros for single-digit months (4)
- MM, month as digits with leading zeros for single-digit months (04)
- MMM, abbreviated month as specified by the current locale ("Nov" in English)
- MMMM, month as specified by the current locale ("November" for English)

The following table defines the format types used to represent years:

- y, year represented only by the last digit (3)
- yy, year represented only by the last two digits. A leading zero is added for single-digit years (03)
- yyy, year represented by a full four or five digits, depending on the calendar used. Thai Buddhist and Korean calendars have five-digit years. The "yyyy" pattern shows five digits for these two calendars, and four digits for all other supported calendars. Calendars that have single-digit or two-digit years, such as for the Japanese Emperor era, are represented differently. A single-digit year is represented with a leading zero, for example, "03". A two-digit year is represented with two digits, for example, "13". No additional leading zeros are displayed.
- yyyy, behaves identically to "yyyy"

The following table defines the format types used to represent era:

- g, period/era string formatted as specified by the CAL\_SERASTRING value (ignored if there is no associated era or period string)
- gg, period/era string formatted as specified by the CAL\_SERASTRING value (ignored if there is no associated era or period string)

The following table defines the format types used to represent hours:

- h, hours with no leading zero for single-digit hours; 12-hour clock
- hh, hours with leading zero for single-digit hours; 12-hour clock
- H, hours with no leading zero for single-digit hours; 24-hour clock

- HH, hours with leading zero for single-digit hours; 24-hour clock

The following table defines the format types used to represent minutes:

- m, minutes with no leading zero for single-digit minutes
- mm, minutes with leading zero for single-digit minutes

The following table defines the format types used to represent seconds:

- s, seconds with no leading zero for single-digit seconds
- ss, seconds with leading zero for single-digit seconds

The following table defines the format types used to represent time markers:

- t, one character time marker string, such as A or P
- tt, multi-character time marker string, such as AM or PM

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