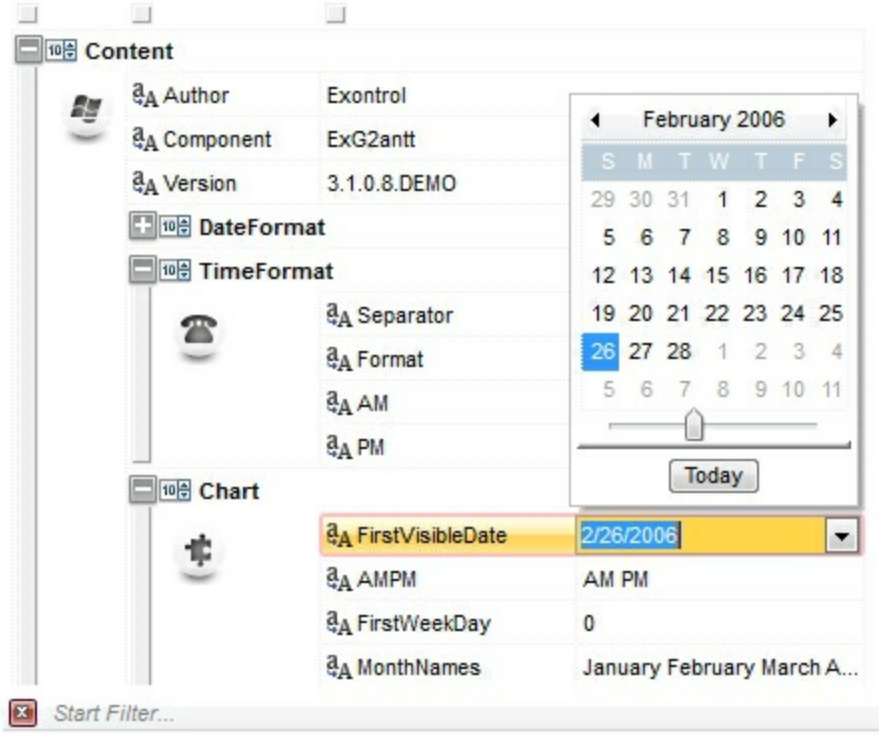
 EXMLGrid

Exontrols new eXMLGrid control provides an innovative grid view look and handles data in XML fashion way. It provides swift and robust performance and a wide range of formatting features never seen on other grids. The eXMLGrid component can be seen as a generalized tree control that allows resizing the node's indentation at runtime.

Features include:

- Skinnable Interface support (ability to apply a skin to any background part)
- Easy way to define the control's visual appearance in design mode, using XP-Theme elements or EBN objects
- Print and Print Preview support
- WYSWYG Template/Layout Editor support.
- Built-in LoadXML and SaveXML methods
- OLE Drag and Drop Support
- Any node supports Built-in HTML format
- Unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the node's background
- Incremental Search support
- Filter-Prompt support, allows you to filter the nodes as you type while the filter bar is visible on the bottom part of the control area
- Editors support: mask, drop down list, check box list, memo fields, spin, OLE Object viewer, color, buttons, sliders, progress bars and more
- ActiveX editors support
- ExpandBar support.
- Single or Multiple Selection support
- Semi-Transparent Selection support
- Muliple Lines HTML Tooltip support
- Picture support
- Mouse wheel support



How to get support?

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

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

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

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

Regards,
Exontrol Development Team

https://www.exontrol.com

constants AlignmentEnum

Specifies the object's alignment.

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

constants AppearanceEnum

Specifies the source's appearance.

Name	Value	Description
None2	0	The source has no borders.
Flat	1	Flat border
Sunken	2	Sunken border
Raised	3	Raised border
Etched	4	Etched border
Bump	5	Bump border

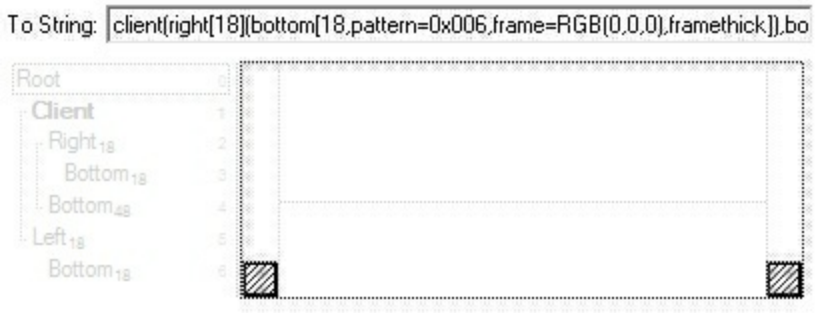
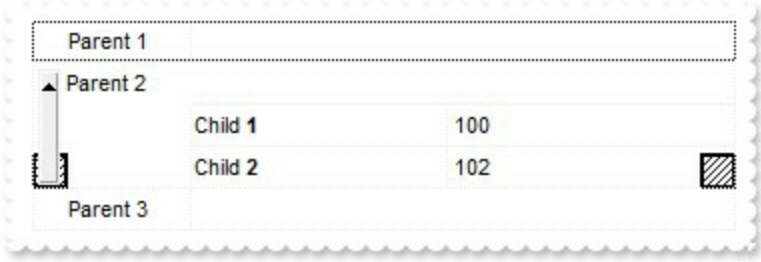
constants AutoSearchEnum

Specifies the type of incremental searching that control performs. Use the AutoSearch property to specify whether the control support incremental searching. The AutoSearch property specifies the kind of searching while user types characters within the control.

Name	Value	Description
exStartWith	-1	Defines the 'starts with' incremental search within the control. If the user type characters within the control the control looks for nodes that start with the typed characters.
exNoAutoSearch	0	The control doesn't support incremental searching.
exContains	1	Defines the 'contains' incremental search within the control. If the user type characters within the control it looks for nodes that contain the typed characters.
exAnyStartWith	2	exAnyStartWith
exAnyContains	3	exAnyContains
exValueStartWith	4	exValueStartWith
exValueContains	5	exValueContains

constants BackgroundExtPropertyEnum

The BackgroundExtPropertyEnum type specifies the UI properties of the part of the EBN you can access/change at runtime. The BackgroundExt property specifies the EBN String format to be displayed on the node'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
exToStringExt	0	Specifies the part's ToString representation. The BackgroundExt 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:*  *where it is applied to an object it looks as follows:*  *(String expression, read-only).*
		Indicates the background color / EBN color to be

exBackColorExt	1	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 the half of the parent object, and so when the parent is resized the client is resized*

exClientExt	2	*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:* • ** ... ** displays the text in **bold** • ** ... ** displays the text in *italics* • **<u> ... </u>** underlines the text • **<s> ... </s>** ~~Strike~~-through text • **<a id;options> ... ** displays an anchor element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link.The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break

exTextExt	4	• **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels. • **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used. • **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold • **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript"

displays the text such as: Text with $_{subscript}$
The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}

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

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

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

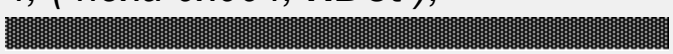
		used to define the height of the font. For instance the "<**sha**>shadow</**sha**>" generates the following picture:  or "*<**sha** 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></**sha**>*" gets:  *(String expression)*
exTextExtWordWrap	5	Specifies that the object is wrapping the text. The 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. *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.*

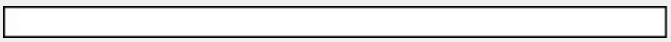
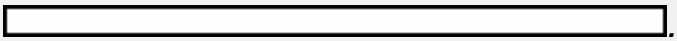
- *17, (hexa 0x11, **Middle-Center**), Text is vertically aligned in the middle, and horizontally aligned at the center.*
- *18, (hexa 0x12, **Middle-Right**), Text is vertically aligned in the middle, and horizontally aligned on the right.*
- *32, (hexa 0x20, **Bottom-Left**), Text is vertically aligned 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**),*  • *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 BackgroundExtStateEnum

The BackgroundExtStateEnum type specifies when the BackgroundExt / BackgroundExtValue property can be applied. The BackgroundExt property specifies the EBN String format to be displayed on the node's background. The BackgroundExtValue property access the value of the giving property for specified part of the EBN. The BackgroundExtStateEnum supports the following values.

Name	Value	Description
exExpandBackgroundExtState	-1	Specifies the BackgroundExt/Value property for the node while it is expanded.
exExpandClipBackgroundExtState	1	Specifies the BackgroundExt/Value property for the node while it is expanded, but clipped to the control's client are.
exCollapseBackgroundExtState	0	Specifies the BackgroundExt/Value property for the node while it is collapsed.
exCollapseClipBackgroundExtState	2	Specifies the BackgroundExt/Value property for the node while it is collapsed, but clipped to the control.

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

If you refer a part of the scroll bar please notice the following:

- All BackgroundPartEnum expressions that starts with **exVS** changes a part in a vertical scroll bar
- All BackgroundPartEnum expressions that starts with **exHS** changes a part in the horizontal scroll bar
- Any BackgroundPartEnum expression that ends with **P** (and starts with exVS or exHS) specifies a part of the scrollbar when it is pressed.
- Any BackgroundPartEnum expression that ends with **D** (and starts with exVS or exHS) specifies a part of the scrollbar when it is disabled.
- Any BackgroundPartEnum expression that ends with **H** (and starts with exVS or exHS) specifies a part of the scrollbar when the cursor hovers it.
- Any BackgroundPartEnum expression that ends with no **H**, **P** or **D** (and starts with exVS or exHS) specifies a part of the scrollbar on normal state.

Name	Value	Description
exExpandButtonUp	0	Specifies the visual appearance for the expand button, when it is up.
exExpandButtonDown	1	Specifies the visual appearance for the expand button, when it is down.
exExpandBarButtonUp	2	Specifies the visual appearance for the button in the control's expand bar, when it is up.
exExpandBarButtonDown	3	Specifies the visual appearance for the button in the control's expand bar, when it is down.
exDropDownButtonUp	4	Specifies the visual appearance for the drop down button, when it is up.
exDropDownButtonDown	5	Specifies the visual appearance for the drop down button, when it is down.
exButtonUp	6	Specifies the visual appearance for the button inside the editor, when it is up. Use the AddButton method to add new buttons to the editor.

exButtonDown	7	Specifies the visual appearance for the button inside the editor, when it is down.
exDateHeader	8	Specifies the visual appearance for the header in a calendar control.
exDateTodayUp	9	Specifies the visual appearance for the today button in a calendar control, when it is up.
exDateTodayDown	10	Specifies the visual appearance for the today button in a calendar control, when it is down.
exDateScrollThumb	11	Specifies the visual appearance for the scrolling thumb in a calendar control.
exDateScrollRange	12	Specifies the visual appearance for the scrolling range in a calendar control.
exDateSeparatorBar	13	Specifies the visual appearance for the separator bar in a calendar control.
exDateSelect	14	Specifies the visual appearance for the selected date in a calendar control.
exSliderRange	15	Specifies the visual appearance for the slider's bar.
exSliderThumb	16	Specifies the visual appearance for the thumb of the slider.
exShowFocusRect	19	exShowFocusRect. Specifies the visual appearance to display the node with the focus.
exSpinUpButtonUp	22	Specifies the visual appearance for the up spin button when it is not pressed.
exSpinUpButtonDown	23	Specifies the visual appearance for the up spin button when it is pressed.
exSpinDownButtonUp	24	Specifies the visual appearance for the down spin button when it is not pressed.
exSpinDownButtonDown	25	Specifies the visual appearance for the down spin button when it is pressed.
exCollapseButtonUp	62	exCollapseButtonUp. Specifies the visual appearance for the expand button, when it is up, and the node is collapsed.
exFooterFilterBarButton	63	exFooterFilterBarButton. Specifies the background color for the closing button in the filter bar.
		Indicates the visual appearance of the borders of the tooltips. Use the ToolTipPopDelay property

exToolTipAppearance	64	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. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ShowToolTip method to display a custom tooltip.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.
exVSUp	256	The up button in normal state.
exVSUpP	257	The up button when it is pressed.
exVSUpD	258	The up button when it is disabled.
exVSUpH	259	The up button when the cursor hovers it.
exVSThumb	260	The thumb part (exThumbPart) in normal state.
exVSThumbP	261	The thumb part (exThumbPart) when it is pressed.
exVSThumbD	262	The thumb part (exThumbPart) when it is disabled.
exVSThumbH	263	The thumb part (exThumbPart) when cursor hovers it.
exVSDown	264	The down button in normal state.
exVSDownP	265	The down button when it is pressed.
exVSDownD	266	The down button when it is disabled.
exVSDownH	267	The down button when the cursor hovers it.
exVSLower	268	The lower part (exLowerBackPart) in normal state.
exVSLowerP	269	The lower part (exLowerBackPart) when it is pressed.
exVSLowerD	270	The lower part (exLowerBackPart) when it is disabled.
exVSLowerH	271	The lower part (exLowerBackPart) when the cursor hovers it.
exVSUpper	272	The upper part (exUpperBackPart) in normal state.
exVSUpperP	273	The upper part (exUpperBackPart) when it is pressed.

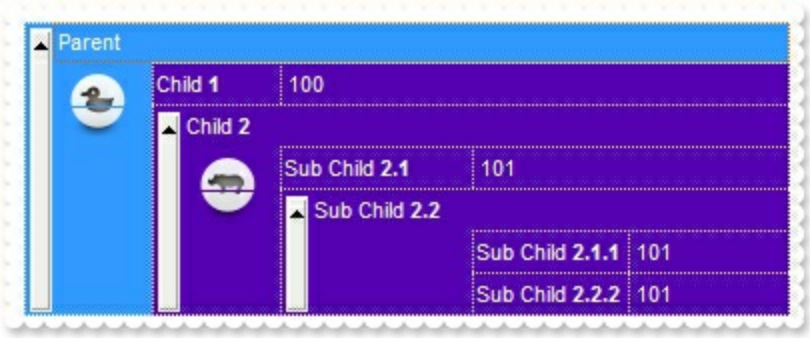
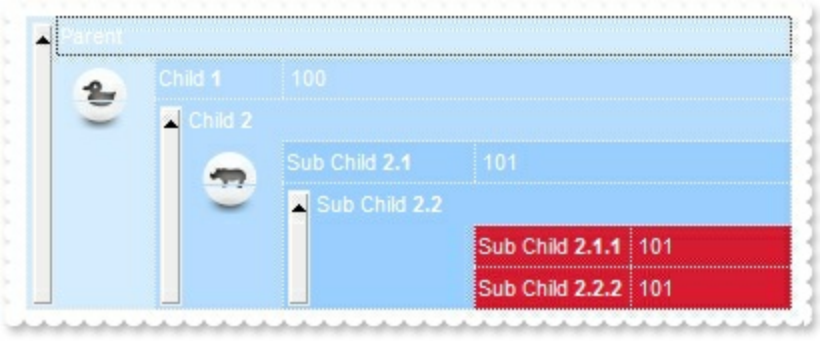
exVSUpperD	274	The upper part (exUpperBackPart) when it is disabled.
exVSUpperH	275	The upper part (exUpperBackPart) when the cursor hovers it.
exVSBack	276	The background part (exLowerBackPart and exUpperBackPart) in normal state.
exVSBackP	277	The background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exVSBackD	278	The background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exVSBackH	279	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exHSLeft	384	The left button in normal state.
exHSLeftP	385	The left button when it is pressed.
exHSLeftD	386	The left button when it is disabled.
exHSLeftH	387	The left button when the cursor hovers it.
exHSThumb	388	The thumb part (exThumbPart) in normal state.
exHSThumbP	389	The thumb part (exThumbPart) when it is pressed.
exHSThumbD	390	The thumb part (exThumbPart) when it is disabled.
exHSThumbH	391	The thumb part (exThumbPart) when the cursor hovers it.
exHSRight	392	The right button in normal state.
exHSRightP	393	The right button when it is pressed.
exHSRightD	394	The right button when it is disabled.
exHSRightH	395	The right button when the cursor hovers it.
exHSLower	396	The lower part (exLowerBackPart) in normal state.
exHSLowerP	397	The lower part (exLowerBackPart) when it is pressed.
exHSLowerD	398	The lower part (exLowerBackPart) when it is disabled.
exHSLowerH	399	The lower part (exLowerBackPart) when the cursor hovers it.
exHSUpper	400	The upper part (exUpperBackPart) in normal state.
		The upper part (exUpperBackPart) when it is

exHSUpperP	401	pressed.
exHSUpperD	402	The upper part (exUpperBackPart) when it is disabled.
exHSUpperH	403	The upper part (exUpperBackPart) when the cursor hovers it.
exHSBack	404	The background part (exLowerBackPart and exUpperBackPart) in normal state.
exHSBackP	405	The background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exHSBackD	406	The background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exHSBackH	407	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exSBtn	324	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), in normal state.
exSBtnP	325	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is pressed.
exSBtnD	326	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is disabled.
exSBtnH	327	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when the cursor hovers it .
exScrollHoverAll	500	Enables or disables the hover-all feature. By default (Background(exScrollHoverAll) = 0), the left/top, right/bottom and thumb parts of the control' scrollbars are displayed in hover state while the cursor hovers any part of the scroll bar (hover-all feature). The hover-all feature is available on Windows 11 or greater, if only left/top, right/bottom, thumb, lower and upper-background parts of the scrollbar are visible, no custom visual-appearance is applied to any visible part. The hover-all feature is always on If Background(exScrollHoverAll) = -1. The Background(exScrollHoverAll) = 1 disables the hover-all feature.
exVSThumbExt	503	exVSThumbExt. The thumb-extension part in normal state.
exVSThumbExtP	504	exVSThumbExtP. The thumb-extension part when it

		is pressed.
exVSThumbExtD	505	exVSThumbExtD. The thumb-extension part when it is disabled.
exVSThumbExtH	506	exVSThumbExtH. The thumb-extension when the cursor hovers it.
exHSThumbExt	507	exHSThumbExt. The thumb-extension in normal state.
exHSThumbExtP	508	exHSThumbExtP. The thumb-extension when it is pressed.
exHSThumbExtD	509	exHSThumbExtD. The thumb-extension when it is disabled.
exHSThumbExtH	510	exHSThumbExtH. The thumb-extension when the cursor hovers it.
exScrollSizeGrip	511	Specifies the visual appearance of the control's size grip, when both scroll bars are displayed.

constants BackModeEnum

The BackModeEnum type specifies the way the control displays the selected nodes. The SelBackMode property specifies the way the control displays the selected nodes. The SingleSel property specifies whether the control supports single or multiple nodes. The BackModeEnum type supports the following values:

Name	Value	Description
exOpaque	0	The selected node overrides the node's background, like color, pictures, and so on. 
exTransparent	1	The background of selected node, is shown as semi-transparent on the node's background. 

constants CheckStateEnum

Specifies the node's state.

Name	Value	Description
Unchecked	0	The node is not checked.
Checked	1	The node is checked.
PartialChecked	2	The node is partially checked.

constants EditorOptionEnum

Specifies different options for a built-in editor. The Option property specifies the editor's options.

Name	Value	Description
exMemoHScrollBar	1	Adds the horizontal scroll bar to a MemoType or MemoDropDownType editor. By default, the Editor.Option(exMemoHScrollBar) is False. (boolean expression)
exMemoVScrollBar	2	Adds the vertical scroll bar to a MemoType or MemoDropDownType editor. By default, the Editor.Option(exMemoVScrollBar) is False. (boolean expression)
exMemoAutoSize	3	Specifies whether the MemoType editor is resized when user alters the text. By default, the Editor.Option(exMemoAutoSize) is True. (boolean expression)
exColorListShowName	4	Specifies whether a ColorListType editor displays the name of the color. By default, the Editor.Option(exColorListShowName) is False. (boolean expression)
exColorShowPalette	5	Specifies whether the ColorList editor displays the palette colors list. By default, the Editor.Option(exColorShowPalette) is True. (boolean expression)
exColorShowSystem	6	Specifies whether the ColorType editor shows the system colors list. By default, the Editor.Option(exColorShowSystem) is True. (boolean expression)
exMemoDropDownWidth	7	Specifies the width for a MemoDropDownType editor. (long expression)
exMemoDropDownHeight	8	Specifies the height for a MemoDropDownType editor. (long expression)
exMemoDropDownAcceptReturn	9	exMemoDropDownAcceptReturn. Specifies whether the Return key is used to add new lines into a MemoDropDownType editor.
exEditRight	10	Right-aligns text in a single-line or multiline edit control. (boolean expression)

exProgressBarBackColor	11	Specifies the background color for a progress bar editor. (color expression)
exProgressBarAlignment	12	Specifies the alignment of the caption inside of a progress bar editor. (AlignmentEnum expression)
exProgressBarMarkTicker	13	Retrieves or sets a value that indicates whether the ticker of a progress bar editor is visible or hidden. (boolean expression)
exDateAllowNullDate	14	Allows you to specify an empty date to a DateType editor. (boolean expression)
exEditPassword	18	Specifies a value that indicates whether an edit control displays all characters as an asterisk (*) as they are typed (passwords). (boolean expression)
exEditPasswordChar	19	Specifies a value that indicates the password character. (character expression)
exLeftArrow	20	(VK_LEFT) Specifies whether the left arrow key is handled by the control or by the current editor. By default, the Option(exLeftArrow) property is True. Use the exLeftArrow option to disable focusing a new cell if the user presses the left arrow key while editing. The option is valid for all editors. (boolean expression)
exRightArrow	21	(VK_RIGHT) Specifies whether the right arrow key is handled by the control or by the current editor. By default, the Option(exRightArrow) property is True. Use the exRightArrow option to disable focusing a new cell if the user presses the right arrow key while editing. The option is valid for all editors. (boolean expression)
exUpArrow	22	(VK_UP) Specifies whether the up arrow key is handled by the control or by the current editor. By default, the Option(exUpArrow) property is True. Use the exUpArrow option to disable focusing a new cell if the user presses the up arrow key while editing. The option is valid for all editors. (boolean expression)
exDownArrow	23	(VK_DOWN) Specifies whether the down arrow key is handled by the control or by the current editor. By default, the Option(exDownArrow) property is True. Use the exDownArrow option to disable focusing a new cell if the user presses the down arrow key

		while editing. The option is valid for all editors.
exHomeKey	24	(VK_HOME) Specifies whether the home key is handled by the control or by the current editor. By default, the Option(exHomeKey) property is True. Use the exHomeKey option to disable focusing a new cell if the user presses the home key while editing. The option is valid for all editors. (boolean expression)
exEndKey	25	(VK_END) Specifies whether the end key is handled by the control or by the current editor. By default, the Option(exEndKey) property is True. Use the exEndKey option to disable focusing a new cell if the user presses the end key while editing. The option is valid for all editors. (boolean expression)
exPageUpKey	26	(VK_PRIOR) Specifies whether the page up key is handled by the control or by the current editor. By default, the Option(exPageUpKey) property is True. Use the exPageUpKey option to disable focusing a new cell if the user presses the page up key while editing. The option is valid for all editors. (boolean expression)
exPageDownKey	27	(VK_NEXT) Specifies whether the page down key is handled by the control or by the current editor. By default, the Option(exPageDownKey) property is True. Use the exPageDownKey option to disable focusing a new cell if the user presses the page down key while editing. The option is valid for all editors. (boolean expression)
exDropDownImage	28	Displays the predefined icon in the control's cell, if the user selects an item from a drop down editor. By default, the exDropDownImage property is True. The option is valid for DropDownListType, PickEdit and ColorListType editors. (boolean expression)
exDateTodayCaption	29	Specifies the caption for the 'Today' button in a DateType editor. By default, the Editor.Option(exDateTodayCaption) is "Today". (string expression)
		Specifies the name for months to be displayed in a DateType editor. The list of months should be delimitated by spaces. By default, the

exDateMonths	30	Editor.Option(exDateMonths) = "January February March April May June July August September October November December". (string expression)
exDateWeekDays	31	Specifies the shortcut for the weekdays to be displayed in a DateType editor. The list of shortcut for the weekdays should be separated by spaces. By default, the Editor.Option(exDateWeekDays) = "S M T W T F S". The first shortcut in the list indicates the shortcut for the Sunday, the second shortcut indicates the shortcut for Monday, and so on. (string expression)
exDateFirstWeekDay	32	Specifies the first day of the week in a DateType editor. By default, the Editor.Option(exDateFirstWeekDay) = 0. The valid values for the Editor.Option(exDateFirstWeekDay) property are like follows: 0 - Sunday, 1 - Monday, 2 - Tuesday, 3 - Wednesday, 4 - Thursday, 5 - Friday and 6 - Saturday. (long expression, valid values are 0 to 6)
exDateShowTodayButton	33	Specifies whether the 'Today' button is visible or hidden in a DateType editor. By default, the Editor.Option(exDateShowTodayButton) property is True. (boolean expression)
exDateMarkToday	34	Gets or sets a value that indicates whether the today date is marked in a DateType editor. By default, Editor.Option(exDateMarkToday) property is False. (boolean expression)
exDateShowScroll	35	Specifies whether the years scroll bar is visible or hidden in a DateType editor. By default, the Editor.Option(exDateShowScroll) property is True. (boolean expression)
exEditLimitText	36	Limits the length of the text that the user may enter into an edit control. By default, the Editor.Option(exEditLimitText) is zero, and so no limit is applied to the edit control. (long expression)
		Specifies the proposed change when user clicks a spin control. The exSpinStep should be a positive number, else clicking the spin has no effect. By default, the exSpinStep option is 1. Integer or floating points allowed as well. For instance, if the

exSpinStep	40	exSpinStep is 0.01, the proposed change when user clicks the spin is 0.01. If the exSpinStep property is 0, the spin control is hidden (useful if you have a slider control).
exSliderWidth	41	Specifies the width in pixels of the slider control. The exSliderWidth value could be 0, when the slider control is hidden, a positive value that indicates the width in pixels of the slider in the control, a negative number when its absolute value indicates the percent of the cell's size being used by the slider. For instance, Option(exSliderWidth) = 0, hides the slider, Option(exSliderWidth) = 100, shows a slider of 100 pixels width, Option(exSliderWidth) = -50, uses half of the cell's client area to display a slider control. By default the Option(exSliderWidth) property is 64 pixels. Use the exSpinStep to hide the spin control. (long expression)
exSliderStep	42	Specifies the proposed change when user clicks a spin control. The exSpinStep should be a positive number, else clicking the spin has no effect. By default, the exSpinStep option is 1. Integer or floating points allowed as well. For instance, if the exSpinStep is 0.01, the proposed change when user clicks the spin is 0.01. If the exSpinStep property is 0, the spin control is hidden (useful if you have a slider control).
exSliderMin	43	Specifies the slider's minimum value. (double expression, by default it is 0)
exSliderMax	44	Specifies the slider's maximum value. (double expression, by default it is 100)
exEditDecimalSymbol	46	Specifies the symbol that indicates the decimal values while editing a floating point number. By default, the exEditDecimalSymbol value is the "Decimal symbol" settings as in the Regional Options, in your control panel. Use the exEditDecimaSymbol option to assign a different symbol for floating point numbers, when Numeric property is exFloat. (long expression, that indicates the ASCII code for the character being used as decimal symbol.)

exDateWeeksHeader	47	Sets or gets a value that indicates whether the weeks header is visible or hidden in a DateType editor. By default, Editor.Option(exDateWeeksHeader) property is False. (boolean expression).
exSliderTickFrequency	53	Gets or sets the interval between tick marks slider types. By default, the exSliderTickFrequency property is 0 which makes the slider to display no ticks. The exSliderTickFrequency property specifies the frequency to display ticks on a slider control. The exSliderStep proposed change in the slider control's position. The exSliderMin and exSliderMax determines the range of values for the slider control. The exSliderWidth option specifies the width of the slider within the cell. (double expression, by default it is 0)
exCalcExecuteKeys	100	Specifies whether the calculator editor executes the keys while focused and the drop down portion is hidden. (boolean expression, by default it is True).
exCalcCannotDivideByZero	101	Specifies the message whether a division by zero occurs in a calendar editor. (string expression, by default it is "Cannot divide by zero.").
exCalcButtonWidth	102	Specifies the width in pixels of the buttons in the calculator editor. (long expression, by default it is 24).
exCalcButtonHeight	103	Specifies the height in pixels of the buttons in the calculator editor. (long expression, by default it is 24).
exCalcButtons	104	Specifies buttons in a calendar editor. The property specifies the buttons and the layout of the buttons in the control. A string expression that indicates the list of buttons being displayed. The rows are separated by chr(13)+chr(10) (vbCrLf) sequence, and the buttons inside the row are separated by ';' character. (string expression)
exCalcPictureUp	105	Specifies the picture when the button is up in a drop down calendar editor. A Picture object that indicates the node's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds

		a picture object. Use the eximages tool to save your picture as base64 encoded format.
exCalcPictureDown	106	Specifies the picture when the button is down in a drop down calendar editor. A Picture object that indicates the node's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format.
exEditAllowOverType	200	Specifies whether the editor supports overtype mode. The option is valid for EditType and MemoType editors. (boolean expression, by default it is False).
exEditOverType	201	Retrieves or sets a value that indicates whether the editor is in insert or overtype mode. The option is valid for EditType and MemoType editors. (boolean expression, by default it is False).

constants EditTypeEnum

Specifies the type of editors being supported by the control. Use the Add method to add a new editor of a specified type. Use the EditType property to change the editor's type. Use the Editor property to assign an editor to a node. Use the Option property to assign different options for a given editor. The Exontrol's EXMLGrid component supports the following type of editors:

Name	Value	Description
ReadOnly	0	Read only editor.

EditType	1	A standard text edit field 
		
		The editor supports the options like:
		• exEditRight, Right-aligns text in a single-line or multiline edit control.
		• exEditPassword, Specifies a value that indicates whether an edit control displays all characters as an asterisk (*) as they are typed (passwords).
		• exEditPasswordChar, Specifies a value that indicates the password character.
		
		The following sample adds an editor of edit type:
		`With XMLGrid1`
		`  With .Editors`
		`  .Add "Edit", EXMLGRIDLibCtl.EditTypeEnum.EditType`
		`  End With`
		`End With`

		 It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window, but it accepts new values at runtime too. The DropDownType editor has associated a standard text edit field too. Use AddItem method to add predefined values to the drop down list. Use the InsertItem method to insert child items to the editor's predefined list. The

DropDownType	2	DropDownRows property specifies the maximum number of visible rows into the drop=down list. The node displays the node's Value or node's Name property.
		
		The following sample adds an editor of drop down type:
		
		``
		With XMLGrid1
		  With .Editors
		  With .Add("DropDownType", EXMLGRIDLibCtl.EditTypeEnum.DropDownType)
		  .AddItem 0, "DHL"
		  .AddItem 1, "Federal Express"
		  .AddItem 2, "Speedy Express"
		  End With
		  End With
		End With
		``

DropDownListType	3	It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. The DropDownListType editor has no standard edit field associated. Use AddItem method to add predefined values to the drop down list. The DropDownRows property specifies the maximum number of visible rows into the drop down list. The item's icon is also displayed if it exists.
		
		The following sample adds a drop down list editor:
		
		``
		With XMLGrid1
		  With .Editors
		  With .Add("DropDownListType", EXMLGRIDLibCtl.EditTypeEnum.DropDownListType)
		
		  .AddItem 0, "DHL"
		  .AddItem 1, "Federal Express"
		  .AddItem 2, "Speedy Express"
		  End With
		``

```
    End With
End With
```

The editor supports the following options:

- exDropDownImage, Displays the predefined icon in the control's cell, if the user selects an item from a drop down editor.

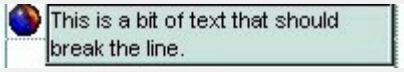
SpinType	4

The SpinType allows your users to view and change numeric values using a familiar up/down button (spin control) combination. The AddItem method has no effect, if the EditType is Spin. Use the exSpinStep option to specify the proposed change when user clicks the spin.

The following sample adds a spin type editor:

```
With XMLGrid1
   With .Editors
      With .Add("Spin",
EXMLGRIDLibCtl.EditTypeEnum.SpinType)
         .Option(exSpinStep) = 10
      End With
   End With
End With
```

MemoType	5



The MemoType is designed to provide a unique and intuitive interface, which you can implement within your application to assist users in working with textual information. If all information does not fit within the edit box, the window of the editor is enlarged. The AddItem method has no effect, if the EditType is Memo.

The following sample adds a memo type editor:

```
With XMLGrid1
   With .Editors
```

```
      With .Add("Memo",
EXMLGRIDLibCtl.EditTypeEnum.MemoType)
         .Appearance = SingleApp
      End With
   End With
End With
```

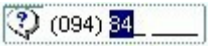
It provides an intuitive interface for your users to check values from pre-defined lists presented in a drop-down window. Each item has a check box associated. The column displays the list of item captions, separated by comma, that is OR combination of the node's value or name. The DropDownRows property specifies the maximum number of visible rows into the drop-down list.

The following sample adds a check list type editor:

CheckListType 6

```
With XMLGrid1
   With .Editors
      With .Add("CheckList",
EXMLGRIDLibCtl.EditTypeEnum.CheckListType)
         .AddItem &H10, "adFldFixed", 4
         .AddItem &H20, "adFldIsNullable", 7
         .AddItem &H80, "adFldLong", 9
         .AddItem &H40, "adFldMayBeNull", 10
         .AddItem &H2, "adFldMayDefer", 3
         .AddItem &H100, "adFldRowID", 5
         .AddItem &H200, "adFldRowVersion", 6
         .AddItem &H8,
"adFldUnknownUpdatable", 7
         .AddItem &H4, "adFldUpdatable", 8
      End With
   End With
End With
```

DateType	7	 The DateType is a date/calendar control (not the Microsoft one). The dropdown calendar provides an efficient and appealing way to edit dates at runtime. The DateType editor has a standard edit control associated. The user can easy select a date by selecting a date from the drop down calendar, or by typing directly the date. The AddItem method has no effect, if the EditType is DateType.
		
		The following sample adds a date editor:
		
		`With XMLGrid1`
		`  With .Editors`
		`  With .Add("Date", EXMLGRIDLibCtl.EditTypeEnum.DateType)`
		`  .Option(exDateMarkToday) = True`
		`  End With`
		`  End With`
		`End With`

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

```
With XMLGrid1
   With .Editors
      With .Add("Phone",
EXMLGRIDLibCtl.EditTypeEnum.MaskType)
         .Mask = "(###) ### - ####"
      End With
   End With
End With
```

ColorType	9	
		You can include a color selection control in your applications via the ColorType editor. Check the ColorListType also. The editor has a standard edit control and a color drop-down window. The color drop-down window contains two tabs that can be used to select colors, the "Pallette" tab shows a grid of colors, while the "System" tab shows the current windows color constants. The AddItem method has no effect, if the EditType is ColorType.
		
		The following simple adds a color type editor:
		
		`With XMLGrid1`
		`  With .Editors`
		`  With .Add("Color",`
		`EXMLGRIDLibCtl.EditTypeEnum.ColorType)`
		`  .Option(exColorShowSystem) = True`
		`  End With`
		`  End With`
		`End With`

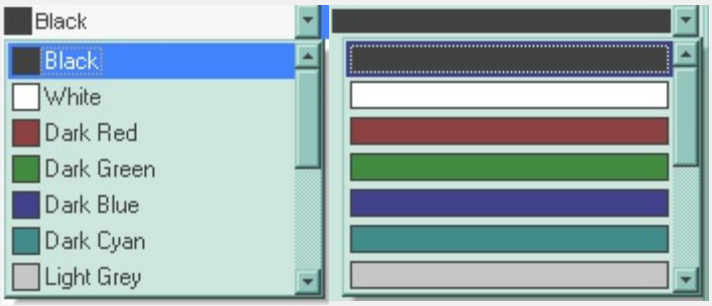
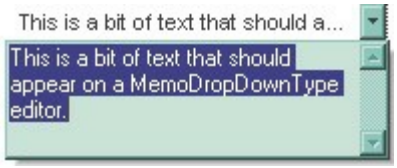
FontType	10	
		Provides an intuitive way for selecting fonts. The FontType editor contains a standard edit control and a font drop-down window. The font drop-down window contains a list with all system fonts. The AddItem method has no effect, if the EditType is FontType. The DropDownRows property specifies the maximum number of visible rows into the drop=down list.
		
		The following sample adds a Font type editor:
		
		`With XMLGrid1`
		`  With .Editors`
		`  .Add "Font", EXMLGRIDLibCtl.EditTypeEnum.FontType`
		`  End With`
		`End With`

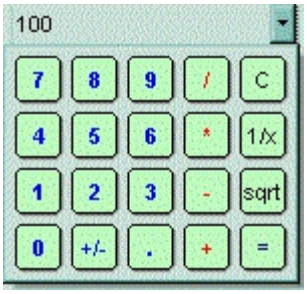
PictureType	11	 The PictureType provides an elegant way for displaying the fields of OLE Object type and cells that have a reference to an IPicture interface. An OLE Object field can contain a picture, a Microsoft Clip Gallery, a package, a chart, PowerPoint slide, a word document, a WordPad documen, a wave file, an so on. In MS Access you can specify the field type to OLE Object. The DropDownMinWidth property specifies the minimum width for the drop=down window. The drop-down window is scaled based on the picture size. The AddItem method has no effect, if the EditType is PictureType.

		The following sample adds a picture type editor:
		
		`With XMLGrid1`
		`  With .Editors`
		`  .Add "Picture", EXMLGRIDLibCtl.EditTypeEnum.PictureType`
		`  End With`
		`End With`

ButtonType	12	The ButtonType editor consists into a  standard edit field and a "..." button. The ButtonClick event is fired if the user has clicked the button. The AddItem method has no effect, if the EditType is ButtonType. Of course, you can apply for multiple buttons using the AddButton method. This is valid no matter for what type of the editor is.
		
		The following sample adds two button editors:
		
		`With XMLGrid1`
		`  With .Editors`
		`  .Add "Button", EXMLGRIDLibCtl.EditTypeEnum.ButtonType`
		`  With .Add("Button2", EXMLGRIDLibCtl.EditTypeEnum.EditType)`
		`  .AddButton "A", 1, 0`
		`  .AddButton "B", 2, 2`
		`  End With`
		`  End With`
		`End With`

ProgressBarType	13	Displays the node's value using a progress bar control.
		It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. The PickEditType editor has a standard edit field associated, that useful for

PickEditType	14	searching items. The DropDownRows property specifies the maximum number of visible rows into the drop=down list. Use AddItem method to add predefined values to the drop down list. The item's icon is also displayed if it exists.
LinkEditType	15	The LinkEditType control allows your application to edit and display hyperlink addresses.
UserEditorType	16	The control is able to use ActiveX controls as a built-in editor. The control uses the UserEditor property to define the user control. If it succeeded the UserEditorObject property retrieves the newly created object. Events like: UserEditOpen, UserEditClose and UserEditorOleEvent are fired when the control uses custom editors.
ColorListType	17	 You can include a color selection control in your applications via the ColorListType editor, also. The editor hosts a predefined list of colors. By default. the following colors are added: Black, White, Dark Red, Dark Green, Dark Yellow, Dark Blue, Dark Magenta, Dark Cyan, Light Grey, Dark Grey, Red, Green, Yellow, Blue, Magenta, Cyan. The AddItem method adds a new color to your color list editor .
		 It provides a multiple lines edit control that's displayed into a drop down window. • The Editor.Option(exMemoDropDownWidth) specifies the width (in pixels) of the MemoDropDownType editor when it is dropped. • The Editor.Option(exMemoDropDownHeight) specifies the height (in pixels) of the MemoDropDownType editor when it is dropped.

MemoDropDownType	18	• The Editor.Option(exMemoDropDownAcceptReturn) specifies whether the user closes the MemoDropDownType editor by pressing the ENTER key. If the Editor.Option(exMemoDropDownAcceptReturn) is True, the user inserts new lines by pressing the ENTER key. The user can close the editor by pressing the CTRL + ENTER key. If the Editor.Option(exMemoDropDownAcceptReturn) is False, the user inserts new lines by pressing the CTRL + ENTER key. The user can close the editor by pressing the ENTER key. • The Editor.Option(exMemoHScrollBar) adds the horizontal scroll bar to a MemoType or MemoDropDownType editor. • The Editor.Option(exMemoVScrollBar) adds the vertical scroll bar to a MemoType or MemoDropDownType editor • Use the Items.CellSingleLine property to specify whether the cell displays multiple lines The AddItem method has no effect, if the EditType is MemoDropDownType.
SliderType	19	 Adds a slider control to a node. Use the exSliderWidth, exSliderStep, exSliderMin, exSliderMax options to control the slider properties. Use the exSpinStep option to hide the spin control.
CalculatorType	20	 Adds a drop down calculator to a node. Use the exCalcExecuteKeys, exCalcCannotDivideByZero, exCalcButtonWidth, exCalcButtonHeight, exCalcButtons, exCalcPictureUp, exCalcPictureDown to specify different options for calculator editor.

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 Files 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
exOLEDropAutomatic	-1	The control triggers the OLE drop events, allowing the programmer to handle the OLE drop operation in code. The control moves the node to a new position when OLE Drag and Drop operation ends.

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

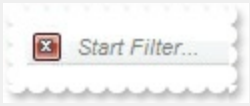
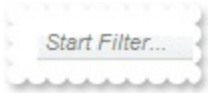
constants ExpandButtonEnum

Specifies the type of +/- (expanding/collapsing) buttons. Use the ExpandButtons property to assign a new type of expanding/collapsing buttons.

Name	Value	Description
exNoButtons	0	No +/-buttons.
exArrow	1	
exPlus	2	
exCustom	3	Use the ExpandButtonsCustom property to assign different icons for +/- buttons.

constants FilterBarVisibleEnum

The FilterBarVisibleEnum type specifies whether the control's filter prompt is shown or hidden. The FilterBarPromptVisible property shows or hides the control's filter prompt. The FilterBarVisibleEnum property supports the following values.

Name	Value	Description
exFilterBarHidden	0	(default)No filter-prompt is shown.
exFilterBarVisible	-1	The filter-prompt is visible until the user clicks the close button. The following screen shows shows the control's filter-prompt, while the FilterBarPromptVisible property is exFilterBarVisible: 
exFilterBarAlwaysVisible	1	The filter-prompt is always visible, so it displays no close button. The following screen shows shows the control's filter-prompt, while the FilterBarPromptVisible property is exFilterBarAlwaysVisible: 

constants FilterPromptEnum

The FilterPromptEnum type specifies the type of prompt filtering. Use the FilterBarPromptType property to specify the type of filtering when using the prompt. The FilterBarPromptPattern property specifies the pattern for filtering. The pattern may contain one or more words being delimited by space characters.

The filter prompt feature supports the following values:

Name	Value	Description
exFilterPromptContainsAll	1	The list includes the items that contains all specified sequences in the filter. Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
exFilterPromptContainsAny	2	The list includes the items that contains any of specified sequences in the filter. Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
exFilterPromptStartWith	3	The list includes the items that starts with any specified sequences in the filter. Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
exFilterPromptEndWith	4	The list includes the items that ends with any specified sequences in the filter. Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
exFilterPromptPattern	16	The filter indicates a pattern that may include wild characters to be used to filter the items in the list. The FilterBarPromptPattern property may include wild characters as follows: • '?' for any single character • '*' for zero or more occurrences of any character • '#' for any digit character • ' ' space delimits the patterns inside the filter

exFilterPromptApplyOnName	32	The filter is applied to name of the node.
exFilterPromptApplyOnValue	64	The filter is applied to the value of the node.
exFilterPromptIncludeChild	128	The filter includes the child nodes for a node that match the criteria.
exFilterPromptCaseSensitive	256	Filtering the list is case sensitive. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.
exFilterPromptStartWords	4608	The list includes the items that starts with specified words, in any position. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.
exFilterPromptEndWords	8704	The list includes the items that ends with specified words, in any position. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.
exFilterPromptWords	12800	The filter indicates a list of words. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.

constants GridLinesEnum

Specifies the type of grid lines. Use the GridLines property to specify whether the control draws the grid lines.

Name	Value	Description
exDotLines	-1	exDotLines. Renders dot grid lines.
exNoGridLines	0	exNoGridLines. No grid lines rendered.
exSolidLines	1	exSolidLines. Renders solid grid lines.

constants HideSelectionEnum

Specifies how the selection is displayed when the control loses the focus. Use the HideSelection property to specify whether the control draws the selection when the control loses the focus.

Name	Value	Description
exHideOnLoseFocus	-1	Hides the selection when the control loses the focus.
exShowAlways	0	exShowAlways. Shows always the selection.
exHideAlways	1	Hides the selection.

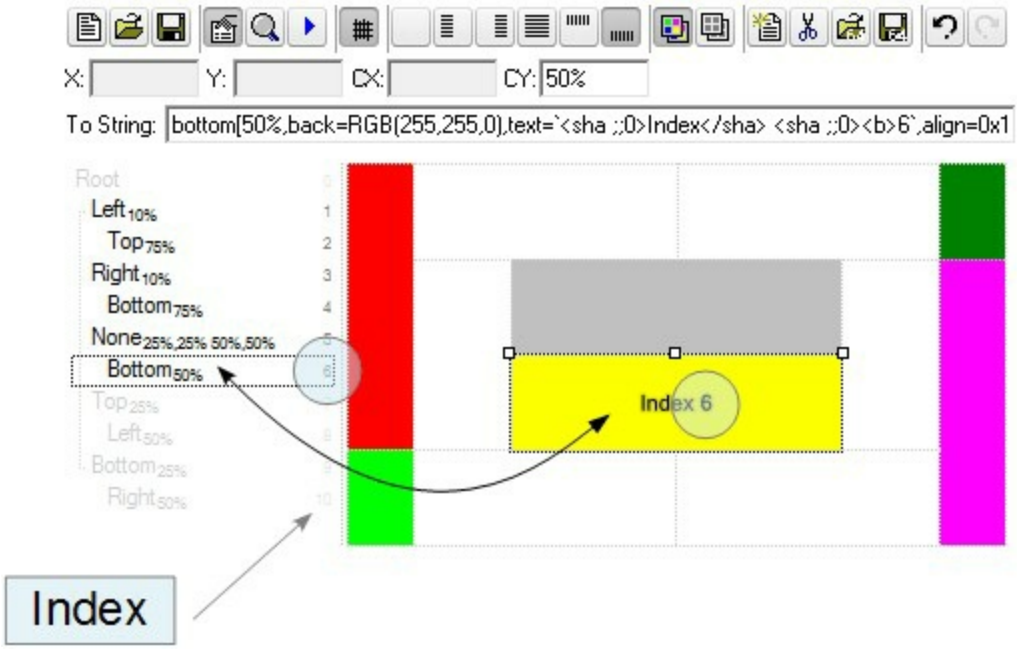
constants HitTestEnum

Specifies the hit test codes supported by the control. Call the HitTest method to determine the location of the specified point relative to the client area of a xml grid view control.

Name	Value	Description
exHTUnknown	0	On the control's client area.
exHTExpandBar	4608	On the control's expand bar area.
exHTClient	4096	On the node's client area.
exHTExpand	4097	On the node's expand/collapse button.
exHTPicture	4098	On the node's picture.
exHTNode	4352	On the node's client area excluding the indent area.
exHTName	4368	On the node's name area.
exHTIcon	4369	On the node's icon area.
exHTText	4370	On the node's caption area.
exHTValue	4384	On the node's value area.
exHTLevelResize	61440	The level border.

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:*

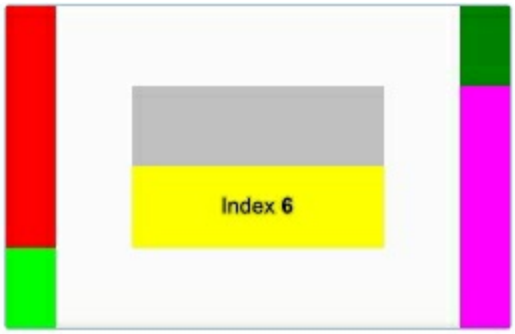


In this sample, there are 11 objects that composes the EBN, so the Index property goes from 0 which indicates the root, and 10, which is the last item in the list

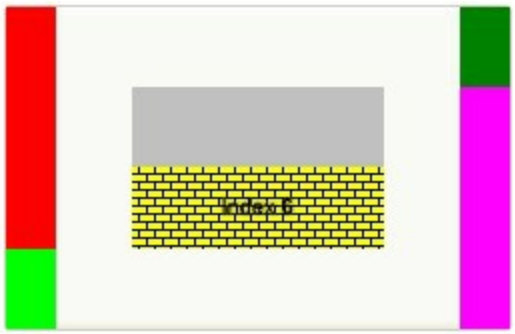
So, let's apply this format to an object, to change the exPatternExt property for the object with the Index 6:

Before calling the BackgroundExt property:

After calling the BackgroundExt 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 InplaceAppearanceEnum

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

Name	Value	Description
NoApp	0	No border
FlatApp	1	Flat appearance
SunkenApp	2	Sunken appearance
RaisedApp	3	Raised appearance
EtchedApp	4	Etched appearance
BumpApp	5	Bump appearance
ShadowApp	6	Shadow appearance
InsetApp	7	Inset appearance
SingleApp	8	Single appearance

constants NumericEnum

Use the Numeric property to specify the format of numbers when editing a node.

Name	Value	Description
exInteger	-1	Allows editing numbers of integer type. The format of the integer number is: **[+/-]digit**, where **digit** is any combination of digit characters.
exAllChars	0	Allows all characters. No filtering.
exFloat	1	Allows editing floating point numbers. The format of the floating point number is: **[+/-]digit[.digit[[e/E/d/D][+/i]digit]]**, where digit is any combination of **digit** characters. Use the exEditDecimalSymbol option to assign a new symbol for '**.**' character (decimal values).

constants PictureDisplayEnum

Specifies how a picture object is displayed. Use the PictureDisplay property to align a picture on the control's background.

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

constants ScrollBarEnum

The ScrollBarEnum type specifies the vertical or horizontal scroll bar in the control. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bars.

Name	Value	Description
exVScroll	0	Indicates the vertical scroll bar.
exHScroll	1	Indicates the horizontal scroll bar.

constants ScrollBarsEnum

Specifies which scroll bars will be visible on a control. Use the ScrollBars property to define which scroll bars are visible.

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

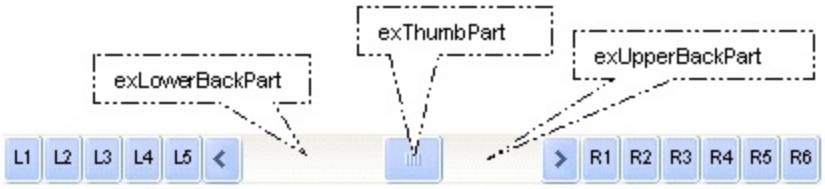
constants ScrollEnum

The ScrollEnum expression indicates the type of scroll that control supports. Use the Scroll method to scroll the control's content by code.

Name	Value	Description
exScrollUp	0	Scrolls up the control by a single node.
exScrollDown	1	Scrolls down the control by a single node.
exScrollVTo	2	Scrolls vertically the control to a specified position.
exScrollLeft	3	Scrolls the control to the left.
exScrollRight	4	Scrolls the control to the right.
exScrollHTo	5	Scrolls horizontaly the control to a specified position

constants ScrollPartEnum

The ScrollPartEnum type defines the parts in the control's scrollbar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollPartCaption property to specify the caption being displayed in any part of the control's scrollbar. The control fires the ScrollButtonClick event when the user clicks any button in the control's scrollbar.

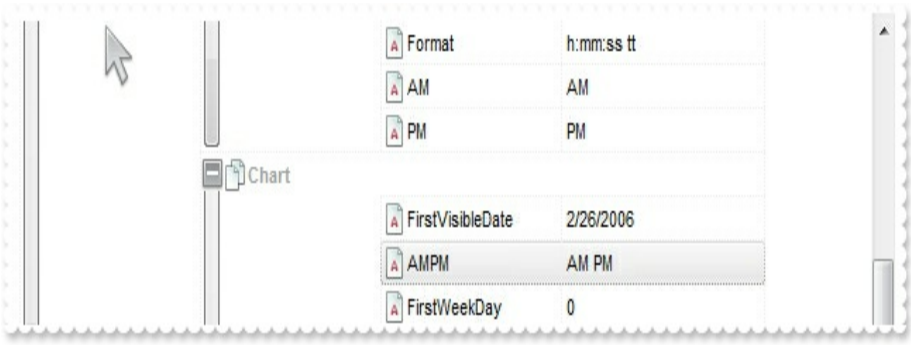
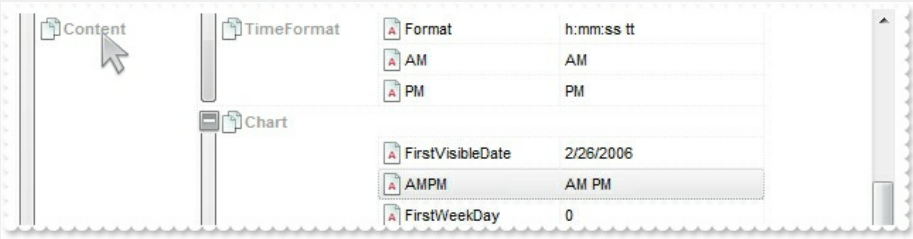
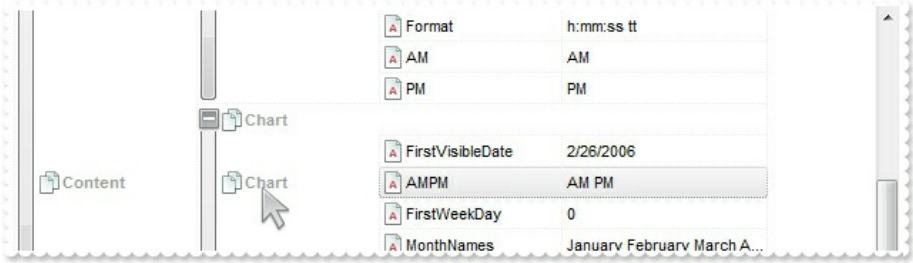


Name	Value	Description
exExtentThumbPart	65536	exExtentThumbPart. The thumb-extension part.
exLeftB1Part	32768	(L1) The first additional button, in the left or top area. By default, this button is hidden.
exLeftB2Part	16384	(L2) The second additional button, in the left or top area. By default, this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

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

constants ShowPartialParentEnum

The ShowPartialParentEnum type specifies whether the control displays the parent node's name on the top or focused node. The ShowPartialParent property specifies where a partial-visible parent shows its content. The ShowPartialParentEnum type supports the following values:

Name	Value	Description
exShowPartialParentHidden	0	No information is shown, like shown bellow: 
exShowPartialParentTop	-1	The parent's content node is shown on the top of the control as you can see on the following picture: 
exShowPartialParentFocus	1	The parent's content node is shown on the focused node as you can see on the following picture: 

constants UIVisualThemeEnum

The UIVisualThemeEnum 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. The Appearance object supports the following properties and methods:

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

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

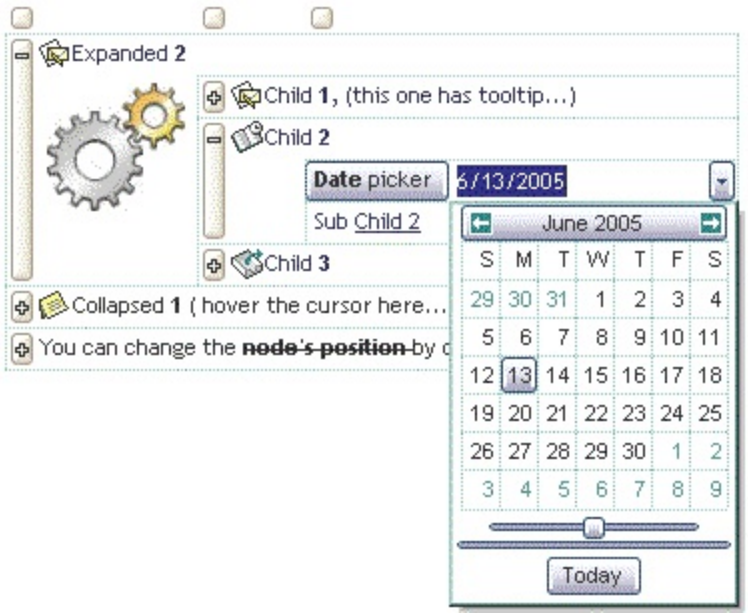
Adds or replaces a skin object to the control.

Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
Skin as Variant	A string expression that indicates: • an Windows XP Theme part, it should start with "XP:". For instance the "**XP:Header 1 2**" indicates the part 1 of the Header class in the state 2, in the current Windows XP theme. 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. • 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 "CP:n l t r b", 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. • 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.



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

high significant byte of the color. For instance, the BackColor = BackColor Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the object that BackColor is applied.

The skin method may change the visual appearance for the following parts in the control:

- up or down expand buttons, Background property
- drop down buttons, buttons in the editors, Background property
- built-in calendar control, Background property
- slider, Background property
- selected nodes, SelBackColor property
- child selected nodes, SelBackColorChild property

The following VB sample changes the visual appearance for the selected node. The SelBackColor property indicates the selection background color. Shortly, we need to add a skin to the Appearance object using the Add method, and we need to set the last 7 bits in the SelBackColor property to indicates the index of the skin that we want to use. The sample applies the "  " to the selected node:

```
With XMLGrid1
   With .VisualAppearance
      .Add &H22, App.Path + "\selected.ebn"
   End With
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = RGB(0,0,255) Or &H22000000
End With
```

The following C++ sample changes the visual appearance for the selected node:

```
#include "Appearance.h"
m_xmlgrid.GetVisualAppearance().Add( 0x22,
COleVariant(_T("D:\\Temp\\EXMLGrid.Help\\selected.ebn")) );
m_xmlgrid.SetSelBackColor( RGB(0,0,255) | 0x22000000 );
m_xmlgrid.SetSelForeColor( 0 );
```

The following VB.NET sample changes the visual appearance for the selected node:

```
With AxXMLGrid1
   With .VisualAppearance
      .Add(&H22, "D:\Temp\EXMLGrid.Help\selected.ebn")
   End With
```

```
    .SelForeColor = Color.Black
    .Template = "SelBackColor = 587137024"
End With
```

where the 587137024 value is the hexa representation of 0x22FF0000

The following C# sample changes the visual appearance for the selected node:

```
axXMLGrid1.VisualAppearance.Add(0x22, "d:\\temp\\EXMLGrid.Help\\selected.ebn");
axXMLGrid1.Template = "SelBackColor = 587137024";
```

where the 587137024 value is the hexa representation of 0x22FF0000.

The following VFP sample changes the visual appearance for the selected node:

```
With thisform.XMLGrid1
    With .VisualAppearance
        .Add(34, "D:\Temp\EXMLGrid.Help\selected.ebn")
    EndWith
    .SelForeColor = RGB(0, 0, 0)
    .SelBackColor = RGB(0,0,255) + 570425344
EndWith
```

The screen shot was generated using the following template:

```
BeginUpdate

    ExpandBarVisible = True
    Font
    {
        Name = "Trebuchet MS"
    }

    Editors
    {
        Add("Edit", 1)
        Add("Slider",19)
        {
            Option(41) = 100
```

```
    }
    Add("Progress",13)
    Add("Calendar",7)
  }

Images("gBJJgBAIFAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl(

  VisualAppearance
  {

Add(1,"gBFLBCJwBAEHhEJAEGg4BWwCg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhU

Add(2,"gBFLBCJwBAEHhEJAEGg4BT4Cg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

Add(3,"gBFLBCJwBAEHhEJAEGg4BV4Fg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

Add(4,"gBFLBCJwBAEHhEJAEGg4BbQFg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

Add(5,"gBFLBCJwBAEHhEJAEGg4BfgFg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAI

  }

  SelBackColor = 67108863
  SelForeColor = RGB(0,0,0)
  BackGround(0) = 16777216
  BackGround(1) = 33554432
  BackGround(2) = 16777216
  BackGround(3) = 33554432
```

```
   BackGround(4) = 67108864
   BackGround(5) = 83886080
   BackGround(8) = 67108863
   BackGround(9) = 67108863
   BackGround(10) = 67108863
   BackGround(11) = 67108863
   BackGround(12) = 67108863
   BackGround(13) = 67108863
   BackGround(14) = 67108863
   BackGround(15) = RGB(208,207,224)
   BackGround(16) = 67108864

   ShowFocusRect = False
   SelBackColorChild = RGB(255,255,255)
   SelForeColorChild = RGB(0,0,0)
   BackColor = RGB(255,255,255)

   OLEDropMode = -1
   LevelWidth(0) = 96
   LevelWidth(1) = 54
   LevelWidth(2) = 72
   ExpandButtons = 2
   Nodes
   {
      Add("Collapsed 1 ( hover the cursor here...)")
      {
         Image = 1
         Picture=
"gBHJJGHA5MJAAEIe4AAAFh0OCERiQbigwEobAsXCAljkcHYwDYQkAli0iGAwHYlCA7HZQIp

         Nodes
         {
            Add("Child 1", "Value 1")
            {
               Image = 2
            }
            Add("Child 2", "Value 2")
```

```
            {
               Image = 3
            }
            Add("Child 3", "Value 3")
            {
               Image = 4
            }
            Add("Child 4", "Value 4")
            {
               Image = 5
            }
         }
      }
      Add("Expanded 2")
      {
         ForeColor = RGB(0,0,80)
         Image = 2
         Picture =
"gBHJJGHA5MJFABAAD3AENhozhpmhqZhrMhr/h0QGcQM0QTMQZkQf8QAESGcSM0STM
         Nodes
         {
            Add("Child 1, (this one has tooltip...)")
            {
               ToolTip = "Exontrol's new eXMLGrid component displays your hierarchical
data using an enhanced grid view, that allows you to provide a new UI to your user.
Exontrol eXMLGrid."
               Image = 2
               Nodes
               {
                  Add("Sub Child 1", "just data")
                  {
                     Editor = "Edit"
                     BackColor = RGB(0,0,255)
                     ForeColor = RGB(255,255,255)
                  }
                  Add("Sub Child 2", "just data")
```

```
            {
               Editor = "Edit"
            }
         }
      }
      Add("Child 2", 2)
      {
'        BackColor = RGB(235,235,245)
'        BackColorChild = RGB(220,220,220)
         ForeColor = RGB(0,0,90)
         Image = 3
         Nodes
         {
            Add(" Date picker", "6/13/2005")
            {
               Editor = "Calendar"
               Selected = True
            }
            Add(" Sub Child 2", 12)
            {
               Editor = "Slider"
            }
         }
         Expanded = True
      }
      Add("Child 3", 2)
      {
'        BackColor = RGB(240,240,240)
'        ForeColor = RGB(0,0,100)
         Image = 4
         Nodes
         {
            Add("Sub Child 1", "just data")
            Add("Sub Child 2", "just data")
         }
      }
   }
```

```
        Expanded = True
      }

      Add("You can change the node's position by clicking the node and dragging it to a
new position.")
      {
        HasChilds = True
        ToolTip = "Exontrol eXMLGrid"
      }
    }
EndUpdate
```

method Appearance.Clear ()

Removes all skins in the control.

Type	Description

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

The skin method may change the visual appearance for the following parts in the control:

- up or down expand buttons, Background property
- drop down buttons, buttons in the editors, Background property
- built-in calendar control, Background property
- slider, Background property
- selected nodes, SelBackColor property
- child selected nodes, SelBackColorChild property

method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

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

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

The skin method may change the visual appearance for the following parts in the control:

- up or down expand buttons, Background property
- drop down buttons, buttons in the editors, Background property
- built-in calendar control, Background property
- slider, Background property
- selected nodes, SelBackColor property
- child selected nodes, SelBackColorChild property

property Appearance.RenderType as Long

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

Type	Description
Long	A long expression that specifies the way EBN object are being applied on the component.

By default, the RenderType property is 0, which indicates that the colors are being applied to EBN object. Imagine that an EBN is just a Color. As you would paint an object using a solid color, the same you can use using the EBN objects. Shortly, all properties or parameters that support EBN objects is indicated in the help file with the description "The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used to paint the ...." A color expression is generally a combination of RRGGBB values, even it is stored in a DWORD value, which means 4 bytes, so actually the color uses only 3 bytes for RR, GG and BB value, (red, green, blue). This way we get one byte that can be used to specify the identifier of the EBN being used. For instance, 0x1000000, indicates the EBN with the identifier 1, since 0x000001, means actually the RGB(0,0,1) which is different. If you need to paint the EBN with a different color, you need to specify the RRGGBB values, as 0x1FF0000 means the EBN object being shown in red as the 0xFF0000 is RGB(255,0,0) which means red, and the 0x1 indicates the EBN with the identifier 1. All controls the support EBN objects provide a VisualAppearance collection. Use the Add method of the VisualAppearance collection to add new EBN objects to the component.

For instance the Color = 0x1000000 indicates that the EBN with the identifier 1, is being displayed on the object's client area. We can use the same EBN object with a different color, by changing the RGB values when setting the Color property such as: Color = 0x1FF0000 applies the EBN with the ID 1, using a Blue as a background color.

The way EBN objects are shown on the objects can be changed using the RenderType property such as follow:

- if **-3 (0xFFFFFFFD)**, *no color* is applied. In this case a *Color = 0x1000000 or Color = 0x1FF0000* displays the same EBN object with no color being applied. In other words, the RGB value in the Color property is ignored or not applied
- if **-2 (0xFFFFFFFD)**, an *OR-color* scheme is used to apply the color on an EBN object. In this case a *Color = 0x1000000 or Color = 0x1FF0000* displays the same EBN object, no color applied for 0x1000000, and 0xFF0000 color being applied on the second. The look or the visual appearance of the second EBN uses the OR-color scheme to show the EBN object with a different color.
- if **-1 (0xFFFFFFFE)**, an *AND-color* scheme is used to apply the color on an EBN object. In this case a *Color = 0x1000000 or Color = 0x1FF0000* displays the same EBN object, no color applied for 0x1000000, and 0xFF0000 color being applied on the

second. The look or the visual appearance of the second EBN uses the AND-color scheme to show the EBN object with a different color.

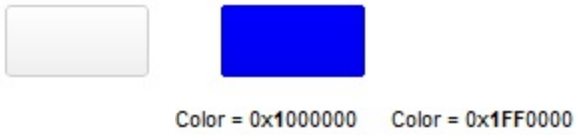
- **(default), 0xTTRRGGBB,** any other value indicates a pair (transparency, color) to be applied to ALL EBN objects. The first significant byte indicates the transparency (a value from 0 to 100), while the other bytes indicates the RGB value. For instance, 0x32FF0000, indicates a transparency of 50% with a blue color, so a 50% blue is begin applied on the EBN to be displayed. This option can be used to apply a color to the entire component, or to show the component with more blue for instance. In other words, the format for RenderType property is AABBGGRR, where the AA could be a value between 0 and 0x64 (100 %) and it indicates the transparency to be applied, and the BBGGRR indicates a RGB color. The RGB color is being applied ONLY if it is not 0x000000 or 0xFFFFFF, In this case the original Color property may indicates the new color to be applied on the EBN object.

The following screen shots shows an EBN object with different type of rendering (RenderType property):

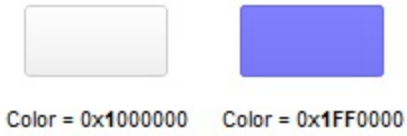
- RenderType = -3,

- RenderType = -2,

- RenderType = -1,



- RenderType = 0x00000000,



- RenderType = 0x32000000,

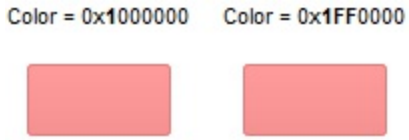
, 50% Transparent (0x32 = 50), Black (0x000000 is 0 or RGB(0,0,0))

- RenderType = 0x64000000,

, 100% Transparent (0x64 = 100), Black (0x000000 is 0 or RGB(0,0,0))

- RenderType = 0x320000FF,

, 50% Transparent (0x32 = 50), Red (0x0000FF is 0 or RGB(255,0,0))



- RenderType = 0x640000FF, , 100% Transparent (0x64 = 100), Red (0x0000FF is 0 or RGB(255,0,0))

Editor object

The Editor object holds information about an editor. Use the Editors property to access the control's editors collection. Use the Add method to add new editors to the Editors collection. Use the Editor property to assign a new editor to the node. The Editor object supports the following properties and methods:

Name	Description
AddButton	Adds a new button to the editor with specified key and aligns it to the left or right side of the editor.
AddItem	Adds a new item to the editor's list.
Appearance	Retrieves or sets the editor's appearance
ButtonWidth	Specifies the width of the buttons in the editor.
ClearButtons	Clears the buttons collection.
ClearItems	Clears the nodes collection.
DropDown	Displays the drop down list.
DropDownAlignment	Retrieves or sets a value that indicates the item's alignment in the editor's drop-down list.
DropDownAutoWidth	Retrieves or sets a value that indicates whether the editor's drop-down window list is automatically computed to fit the entire list.
DropDownMinWidth	Specifies the minimum drop-down list width if the DropDownAutoWidth is False.
DropDownRows	Retrieves or sets a value that indicates the maximum number of visible rows in the editor's drop- down list.
DropDownVisible	Retrieves or sets a value that indicates whether the editor's drop down window is visible or hidden.
EditType	Retrieves or sets a value that indicates the type of the contained editor.
ExpandAll	Expands all nodes in the editor's list.
ExpandItem	Expands or collapses an item in the editor's list.
FindItem	Finds an item given its value or caption.
Index	Gets the editor's index in the editors collection.
InsertItem	Inserts a child item to the editor's list.
ItemToolTip	Gets or sets the text displayed when the mouse pointer hovers over a predefined item.

Key	Specifies the editor's key.
Locked	Determines whether the editor is locked or unlocked.
Mask	Retrieves or sets a value that indicates the mask used by the editor.
MaskChar	Retrieves or sets a value that indicates the character used for masking.
Numeric	Specifies whether the editor enables numeric values only.
Option	Specifies an option for the editor.
PartialCheck	Retrieves or sets a value that indicates whether the associated check box has two or three states.
PopupAppearance	Retrieves or sets a value that indicates the drop-down window's appearance.
RemoveButton	Removes a button given its key.
RemoveItem	Removes an item from the editor's predefined values list.
SortItems	Sorts the list of nodes in the editor.
UserEditor	Specifies the control's identifier and the control's runtime license key when EditType is UserEditor.
UserEditorObject	Gets the user editor object when EditType is UserEditor.

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

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

Type	Description
Key as Variant	A Variant expression that indicates the key of the button being added.
Image as Variant	A long expression that indicates the index of icon being displayed in the button.
Align as Variant	An AlignmentEnum expression that indicates the button's alignment inside the node.
ToolTip as Variant	A string expression that indicates the button's tooltip. The button's tooltip shows up when user hovers the cursor over the button. The ToolTip value accepts built-in HTML format like described in the Remarks paragraf.
ToolTipTitle as Variant	A string expression that indicates the title of the button's tooltip.
ShortcutKey as Variant	A short expression that indicates the shortcut key being used to simulate clicking the button. The lower byte indicates the code of the virtual key, and the higher byte indicates the states for SHIFT, CTRL and ALT keys (last insignificant bits in the higher byte). The ShortcutKey expression could be 256 *((shift ? 1 : 0) + (ctrl ? 2 : 0) + (alt ? 4 : 0)) + vbKeyCode, For instance, a combination like CTRL + F3 is 256 * 2 + vbKeyF3, SHIFT + CTRL + F2 is 256 *(1 + 2) + vbKeyF2, and SHIFT + CTRL + ALT + F5 is 256 * (1 + 2 + 4) + vbKeyF5.

Use the AddButton method to add new buttons to an editor. Use the ButtonClick event to notify your application that the user clicks a button inside a node. Use the Editors property to access the control's Editors collection. Use the Add method to add new type of editors to the control. Use the Editor property to assign an editor to a node. Use the ButtonWidth property to specify the width of the buttons inside the editor.

The following sample displays a message box when user clicks the 'A' button:

```
Private Sub Form_Load()
  With XMLGrid1
```

```
    .BeginUpdate
      With .Editors
        With .Add("Spin")
          .ButtonWidth = 18
          .EditType = SpinType
          .AddButton "A", 1
        End With
      End With
      With .Nodes
        With .Add("Spin", 1)
          .Editor = "Spin"
        End With
      End With
    .EndUpdate
  End With
End Sub

Private Sub XMLGrid1_ButtonClick(ByVal Node As EXMLGRIDLibCtl.INode, ByVal Key As
Variant)
  If Key = "A" Then
    MsgBox "You have clicked the 'A' button."
  End If
End Sub
```

The control uses built-in HTML tags to display the caption using HTML format. The control supports the following HTML tags:

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

the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.

- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the

offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{subscript}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}

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

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

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

Adds a new item to the editor's list.

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

Use the AddItem method to add new items to the editor's predefined list. If the AddItem method uses a Value already defined, the old item is replaced. The AddItem method has effect for the following type of editors: **DropDownType, DropDownListType, PickEditType,** and **CheckListType**. Check each EditType value for what Value argument should contain. Use the RemoveItem method to remove a particular item from the predefined list. Use the ClearItems method to clear the entire list of predefined values. Use the SortItems to sort the items. Use the ItemToolTip property to assign a tooltip to a predefined item into a drop down list. The Caption parameter supports built-in HTML format like follows:

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

the color in hexa values.

- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{subscript}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a

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

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

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

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

property Editor.Appearance as InplaceAppearanceEnum

Retrieves or sets the editor's appearance

Type	Description
InplaceAppearanceEnum	An InplaceAppearanceEnum expression that defines the editor's appearance.

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

property Editor.ButtonWidth as Long

Specifies the width of the buttons in the editor.

Type	Description
Long	A long expression that defines the width of the buttons in the editor, added using the AddButton method.

Use the ButtonWidth property to increase or decrease the width of buttons in the editor. The button's height is the same with the NodeHeight property. If the ButtonWidth property is zero (0), the control hides the buttons. Use the AddButton method to add new buttons to the editor. Use the Editor property to assign an editor to a node.

method Editor.ClearButtons ()

Clears the buttons collection.

Type	Description

Use the ClearButtons method to clear the entire list of buttons added using AddButton method. Use the RemoveButton method to remove a particular button, given its key. Use the ButtonWidth property to hide the buttons.

method Editor.ClearItems ()

Clears the items collection.

Type	Description

The ClearItems method clears the predefined values added using AddItem, InsertItem methods. Use the RemoveItem method to remove a particular item. Use the DropDownVisible property to hide the drop-down window.

method Editor.DropDown ()

Displays the drop down list.

Type	Description

The DropDown method shows the drop down portion of the editor.

property Editor.DropDownAlignment as AlignmentEnum

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

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the item's alignment into the editor's drop-down list.

Use the DropDownAlignment property to align the items in the editor's drop-down list.

property Editor.DropDownAutoWidth as Boolean

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

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

Use the DropDownAutoWidth property to specify when you let the control computes the drop-down list width, or whenever the width is specified by the DropDownMinWidth property

property Editor.DropDownMinWidth as Long

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

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

The DropDownMinWidth property has no effect if the DropDownAutoWidth property is True.

property Editor.DropDownRows as Long

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

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

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

property Editor.DropDownVisible as Boolean

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

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

Use the DropDownVisible property to hide the editor's drop-down window. Use the ButtonWidth property to hide the editor buttons.

property Editor.EditType as EditTypeEnum

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

Type	Description
EditTypeEnum	An EditTypeEnum expression that specifies the type of the editor.

Use the EditType property to specify the type of the editor. Use the Add method to insert new type of editors to the control. You can specify the type of editor at the adding time. Use the AddItem method to insert predefined items to a drop down list editor. Use the Option property to define options for a specific type of editor.

The following sample adds an integer editor and a float point editor:

```
With XMLGrid1
  .BeginUpdate
    .AutoEdit = True
    With .Editors.Add("Float", EditType)
      .Numeric = exFloat
    End With
    With .Editors.Add("Integer", EditType)
      .Numeric = exInteger
    End With
    With .Nodes
      With .Add("<b>Float</b> Number")
        .Editor = "Float"
      End With
      With .Add("<b>Integer</b> Number")
        .Editor = "Integer"
      End With
    End With
  .EndUpdate
End With
```

The following sample adds check list editor:

```
With XMLGrid1
  .BeginUpdate
    .AutoEdit = True
```

```
    With .Editors.Add("CL", CheckListType)
      .AddItem 1, "One"
      .AddItem 2, "Two"
      .AddItem 4, "Four"
    End With
    With .Nodes
      With .Add("Check", 3)
        .Editor = "CL"
      End With
    End With
  .EndUpdate
End With
```

The following sample adds a progress bar editor:

```
With XMLGrid1
  .BeginUpdate
    .AutoEdit = True
    With .Editors.Add("PRO", ProgressBarType)
      .Option(exProgressBarBackColor) = vbGreen
    End With
    With .Nodes
      With .Add("Progress", 34)
        .Editor = "PRO"
      End With
    End With
  .EndUpdate
End With
```

method Editor.ExpandAll ()

Expands all items in the editor's list.

Type	Description

[not supported yet] By default, in your editor items that contain child items are collapsed. Use the ExpandAll method to expand all items in the editor. Use the InsertItem method to insert child items.

property Editor.ExpandItem(Value as Variant) as Boolean

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

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

[not supported yet] By default, the items are collapsed. Use the ExpandItem to expand a specified item. Use the ExpandAll method to expand all items in the editor. Use the InsertItem method to insert a child item to your built-in editor

property Editor.FindItem (Value as Variant) as Variant

Finds an item given its value or caption.

Type	Description
Value as Variant	A long expression that indicates the value of the item being searched, a string expression that indicates the caption of the item being searched.
Variant	A string expression that indicates the caption of the item, if the Value is a long expression, a long expression that indicates the item's value if Value is a string expression.

The FindItem property retrieves an empty (VT_EMPTY) value if no item was found.

property Editor.Index as Long

Gets the editor's index in the editors collection.

Type	Description
Long	A long expression that indicates the index of the editor in the Editors collection.

The Index property specifies the index of the editor in the control's editors collection. The Key property specifies the editor's key.

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

Inserts a child item to the editor's list.

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

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

property Editor.ItemToolTip(Value as Variant) as String

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

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

Use the ItemToolTip property to assign a tooltip for a drop down list value. Use the AddItem or InsertItem methods to insert new items to the drop down predefined list. The ItemToolTip property may include built-in HTML format.

property Editor.Key as Variant

Specifies the editor's key.

Type	Description
Variant	A string or long expression that indicates the key of the editor.

The Key property specifies the editor's key. The Index property specifies the index of the editor in the control's editors collection. The Key property is read only. Use the Add method to add new type of editors with specified keys. Use the Editor property to assign an editor to a node.

property Editor.Locked as Boolean

Determines whether the editor is locked or unlocked.

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

Use the Locked property to lock the editor. If the editor is locked, the user is not able to change the control's content using the editor.

property Editor.Mask as String

Retrieves or sets a value that indicates the mask used by the editor.

Type	Description
String	A string expression that defines the editor's mask.

Use the Mask property to filter characters during data input. Use the Numeric property to filter for numbers.

Use the Mask property to control the entry of many types of formatted information such as telephone numbers, social security numbers, IP addresses, license keys etc. The Mask property has effect for the following edit types: DropDownType, SpinType, DateType, MaskType, FontType, PickEditType. Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. Use the Editing property to check whether the control is running in the edit mode.

Use the MaskChar property to change the masking character. If the Mask property is empty no filter is applied. The Mask property is composed by a combination of regular characters, literal escape characters, and masking characters. The Mask property can contain also alternative characters, or range rules. A literal escape character is preceded by a \ character, and it is used to display a character that is used in masking rules. Here's the list of all rules and masking characters:

Rule	Name	Description
#	Digit	Masks a digit character. [0-9]
x	Hexa Lower	Masks a lower hexa character. [0-9],[a-f]
X	Hexa Upper	Masks a upper hexa character. [0-9],[A-F]
A	AlphaNumeric	Masks a letter or a digit. [0-9], [a-z], [A-Z]
?	Alphabetic	Masks a letter. [a-z],[A-Z]
<	Alphabetic Lower	Masks a lower letter. [a-z]
>	Alphabetic Upper	Masks an upper letter. [A-Z]
*	Any	Mask any combination of characters.
\	Literal Escape	Displays any masking characters. The following combinations are valid: \#,\x,\X,\A,\?,\<,\>,\\,\{,\[
{nMin,nMax}	Range	Masks a number in a range. The nMin and nMax values should be numbers. For instance the mask {0,255} will mask any number between 0 and 255.
		Masks any characters that are contaied by brackets []. For

[...]	Alternative	instance, the [abcA-C] mask any character: a,b,c,A,B,C

The following sample adds an editor for masking phone numbers:

```
With XMLGrid1
   .BeginUpdate
      With .Editors.Add("Phone", MaskType)
         .Mask = "(###) - ### ####"
      End With
      With .Nodes
         With .Add("Phone", "")
            .Editor = "Phone"
         End With
      End With
   .EndUpdate
End With
```

property Editor.MaskChar as Long

Retrieves or sets a value that indicates the character used for masking.

Type	Description
Long	A long expression that indicates the ASCII code for the masking character.

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

property Editor.Numeric as NumericEnum

Specifies whether the editor enables numeric values only.

Type	Description
NumericEnum	A NumericEnum expression that indicates whether integer or floating point numbers are allowed.

The Numeric property has effect only if the editor contains an edit box. Use the Numeric property to add intelligent input filtering for integer, or floating points numbers. Use the exSpinStep option to specify the proposed change when user clicks a spin control, if the cell's editor is of SpinType type. Use the exEditDecimaSymbol option to specify the symbol being used by decimal value while editing a floating point number.

property Editor.Option(Name as EditorOptionEnum) as Variant

Specifies an option for the editor.

Type	Description
Name as EditorOptionEnum	An EditorOptionEnum expression that indicates the editor's option being changed.
Variant	A Variant expression that indicates the value for editor's option

Use the Option property to define options for a certain type of editor.

The following sample adds a password editor:

```
With XMLGrid1
  .BeginUpdate
    With .Editors.Add("Password", EditType)
      .Option(exEditPassword) = True
    End With
    With .Nodes
      With .Add("Password", "")
        .Editor = "Password"
      End With
    End With
  .EndUpdate
End With
```

The following sample specifies that the editor "A" requires all arrow keys. By default, the control uses the arrow key to navigate through the nodes.

```
With XMLGrid1
  .BeginUpdate
    With .Editors.Add("A", EditType)
      .Option(exLeftArrow) = False
      .Option(exRightArrow) = False
      .Option(exUpArrow) = False
      .Option(exDownArrow) = False
      .Option(exHomeKey) = False
      .Option(exEndKey) = False
    End With
```

```
      With .Nodes
         With .Add("Use Arrow Keys", "swssw")
            .Editor = "A"
         End With
      End With
   .EndUpdate
End With
```

property Editor.PartialCheck as Boolean

Retrieves or sets a value that indicates whether the associated check box has two or three states.

Type	Description
Boolean	A boolean expression that indicates whether the associated check box has two or three states.

[not supported yet]

property Editor.PopupAppearance as InplaceAppearanceEnum

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

Type	Description
InplaceAppearanceEnum	An InplaceAppearanceEnum expression that defines the drop-down window's border style.

Use the PopupAppearance property to change the drop-down window's border style. Use the Appearance property to define the editor's appearance.

method Editor.RemoveButton (Key as Variant)

Removes a button given its key.

Type	Description
Key as Variant	A Variant value that determines the button's key being deleted. The Key should be the same as used in the AddButton method.

Use the RemoveButton method to remove a button, given its key. Use the ButtonWidth property to hide the editor buttons. Use the ClearButtons method.

method Editor.RemoveItem (Value as Long)

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

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

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

method Editor.SortItems ([Ascending as Variant], [Reserved as Variant])

Sorts the list of items in the editor.

Type	Description
Ascending as Variant	A boolean expression that indicates the sort order of the items.
Reserved as Variant	For future use only.

Use the SortItems method to sort the items in a drop down editor.

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

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

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

The UserEditor property creates a new type of editor based on the ControlID parameter. Use the UserEditorObject property to access the newly created object. The UserEditorObject property points to nothing if the control wasn't able to create the user editor based on the ControlID. Also, if the user control requires a runtime license key, and the License parameter is empty or doesn't match, the UserEditorObject property points to nothing. The control fires the UserEditorOpen event when a ActiveX editor is about to be opened. The control fires the UserEditorClose event when the user editor needs to be closed. The control fires the UserEditorOleEvent event each time when an user editor fires an event.

The following VB sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
With XMLGrid1.Editors
  With .Add("excombobox", UserEditorType)
    .UserEditor "Exontrol.ComboBox", ""
    With .UserEditorObject
      .BeginUpdate
      .LabelHeight = XMLGrid1.NodeHeight - 3
      .LinesAtRoot = True
      .HeightList = 256
      .WidthList = 256
      .IntegralHeight = True
      .Columns.Add ("Name")
      .Columns.Add ("Value")
      .ColumnAutoResize = True
      With .Items
        Dim h As Long, h1 As Long
```

```
            h = .AddItem("Item 1")
            .CellCaption(h, 1) = "Item 1.2"
            h1 = .InsertItem(h, , "SubItem 1")
            .CellCaption(h1, 1) = "SubItem 1.2"
            h1 = .InsertItem(h, , "SubItem 2")
            .CellCaption(h1, 1) = "SubItem 2.2"
            .ExpandItem(h) = True
          End With
          .EndUpdate
        End With
    End With
End With
```

The following C++ sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
#include "Editor.h"
#include "Editors.h"
COleVariant vtMissing; V_VT( &vtMissing; ) = VT_ERROR;
CEditors editors = m_xmlgrid.GetEditors();
CEditor editor = editors.Add( COleVariant( "excombobox" ), 16 /*UserEditorType*/ );
editor.UserEditor( "Exontrol.ComboBox", "" );
EXCOMBOBOXLib::IComboBoxPtr spComboBox = editor.GetUserEditorObject();
if ( spComboBox != NULL )
{
    spComboBox->BeginUpdate();
    spComboBox->LabelHeight = m_xmlgrid.GetNodeHeight() - 3;
    spComboBox->LinesAtRoot = EXCOMBOBOXLib::exLinesAtRoot;
    spComboBox->put_HeightList( vtMissing, 256 );
    spComboBox->put_WidthList( vtMissing, 256 );
    spComboBox->IntegralHeight = true;
    spComboBox->Columns->Add("Name");
    spComboBox->Columns->Add("Value");
    spComboBox->ColumnAutoResize = true;
    EXCOMBOBOXLib::IItemsPtr spItems = spComboBox->Items;
    long h = spItems->AddItem(COleVariant( "Item 1" ));
    spItems->put_CellCaption(COleVariant(h),COleVariant((long)1), COleVariant( "Item 1.2" )
);
```

```
    long h1 = spItems->InsertItem(h, vtMissing, COleVariant( "SubItem 1") );
    spItems->put_CellCaption(COleVariant(h1),COleVariant((long)1), COleVariant( "SubItem 1.2" ) );
    h1 = spItems->InsertItem(h, vtMissing, COleVariant( "SubItem 2") );
    spItems->put_CellCaption(COleVariant(h1),COleVariant((long)1), COleVariant( "SubItem 2.2" ) );
    spItems->put_ExpandItem(h, true );
    spComboBox->EndUpdate();
}
```

The sample requires the #import <excombobox.dll> to include the ExComboBox's type library. The #import <excombobox.dll> creates EXCOMBOBOXLib namespace that includes all definitions for objects and types that the ExComboBox control exports.

The following VB.NET sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
With AxXMLGrid1.Editors
    With .Add("excombobox", EXMLGRIDLib.EditTypeEnum.UserEditorType)
        .UserEditor("Exontrol.ComboBox", "")
        With .UserEditorObject
            .BeginUpdate()
            .LabelHeight = AxXMLGrid1.NodeHeight - 3
            .LinesAtRoot = True
            .HeightList = 256
            .WidthList = 256
            .IntegralHeight = True
            .Columns.Add("Name")
            .Columns.Add("Value")
            .ColumnAutoResize = True
            With .Items
                Dim h, h1 As Integer
                h = .AddItem("Item 1")
                .CellCaption(h, 1) = "Item 1.2"
                h1 = .InsertItem(h, , "SubItem 1")
                .CellCaption(h1, 1) = "SubItem 1.2"
                h1 = .InsertItem(h, , "SubItem 2")
                .CellCaption(h1, 1) = "SubItem 2.2"
                .ExpandItem(h) = True
```

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

The following C# sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
EXMLGRIDLib.Editor editor = axXMLGrid1.Editors.Add("excombobox",
EXMLGRIDLib.EditTypeEnum.UserEditorType);
editor.UserEditor("Exontrol.ComboBox", "");
EXCOMBOBOXLib.ComboBox comboBox = editor.UserEditorObject as
EXCOMBOBOXLib.ComboBox;
if ( comboBox != null )
{
   comboBox.BeginUpdate();
   comboBox.LabelHeight = axXMLGrid1.NodeHeight - 3;
   comboBox.LinesAtRoot = EXCOMBOBOXLib.LinesAtRootEnum.exLinesAtRoot ;
   comboBox.set_HeightList( null, 256 );
   comboBox.set_WidthList( null, 256 );
   comboBox.IntegralHeight = true;
   comboBox.Columns.Add("Name");
   comboBox.Columns.Add("Value");
   comboBox.ColumnAutoResize = true;
   EXCOMBOBOXLib.Items items = comboBox.Items;
   int h = items.AddItem("Item 1");
   items.set_CellCaption(h, 1, "Item 1.2" );
   int h1 = items.InsertItem(h, null, "SubItem 1");
   items.set_CellCaption(h1, 1,"SubItem 1.2");
   h1 = items.InsertItem(h, null, "SubItem 2");
   items.set_CellCaption(h1, 1,"SubItem 2.2");
   items.set_ExpandItem(h, true);
   comboBox.EndUpdate();
}
```

In C# your project needs a new reference to the Exontrol's ExComboBox control library. Use the Project\Add Reference\COM item to add new reference to a COM object. Once that you added a reference to the Exontrol's ExComboBox the EXCOMBOBOXLib

namespace is created. The EXCOMBOBOXLib namespace contains definitions for all objects that ExComboBox control exports.

The following VFP sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
With thisform.XMLGrid1.Editors
   With .Add("excombobox", 16) && UserEditorType
      .UserEditor("Exontrol.ComboBox", "")
      With .UserEditorObject
         .BeginUpdate
         .LabelHeight = thisform.XMLGrid1.NodeHeight - 3
         .LinesAtRoot = -1
         .HeightList(0) = 256
         .WidthList(0) =256
         .IntegralHeight = .t.
         .Columns.Add ("Name")
         .Columns.Add ("Value")
         .ColumnAutoResize = .t.
         With .Items
            .DefaultItem = .AddItem("Item 1")
            h = .DefaultItem
            .CellCaption(0, 1) = "Item 1.2"
            .DefaultItem = .InsertItem(h, , "SubItem 1")
            .CellCaption(0, 1) = "SubItem 1.2"
            .DefaultItem = .InsertItem(h, , "SubItem 2")
            .CellCaption(0, 1) = "SubItem 2.2"
            .DefaultItem = h
            .ExpandItem(0) = .t.
         EndWith
         .EndUpdate
      EndWith
   EndWith
EndWith
```

property Editor.UserEditorObject as Object

Gets the user editor object when EditType is UserEditor.

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

Use the UserEditorOpen property to access the ActiveX user editor. Use the UserEditor property to initialize the ActiveX user editor. The UserEditorObject property retrieves the ActiveX control created when UserEditor method was invoked. The type of object returned by the UserEditorObject depends on the ControlID parameter of the UserEditor method. For instance, the type of the created object when UserEditor("Exontrol.ComboBox") is used, is EXCOMBOBOXLibCtl.ComboBox. The UserEditorObject property gets nothing if the UserEditor method fails. The control fires the UserEditorOpen event when an user editor is about to be opened. The control fires the UserEditorClose event when the control closes an user editor. The control fires the UserEditorOleEvent event each time when an user editor fires an event.

The following VB sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
With XMLGrid1.Editors
   With .Add("excombobox", UserEditorType)
      .UserEditor "Exontrol.ComboBox", ""
      With .UserEditorObject
         .BeginUpdate
         .LabelHeight = XMLGrid1.NodeHeight - 3
         .LinesAtRoot = True
         .HeightList = 256
         .WidthList = 256
         .IntegralHeight = True
         .Columns.Add ("Name")
         .Columns.Add ("Value")
         .ColumnAutoResize = True
         With .Items
            Dim h As Long, h1 As Long
            h = .AddItem("Item 1")
            .CellCaption(h, 1) = "Item 1.2"
            h1 = .InsertItem(h, , "SubItem 1")
            .CellCaption(h1, 1) = "SubItem 1.2"
            h1 = .InsertItem(h, , "SubItem 2")
```

```
            .CellCaption(h1, 1) = "SubItem 2.2"
            .ExpandItem(h) = True
          End With
          .EndUpdate
        End With
    End With
End With
```

The following C++ sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
#include "Editor.h"
#include "Editors.h"
COleVariant vtMissing; V_VT( &vtMissing; ) = VT_ERROR;
CEditors editors = m_xmlgrid.GetEditors();
CEditor editor = editors.Add( COleVariant( "excombobox" ), 16 /*UserEditorType*/ );
editor.UserEditor( "Exontrol.ComboBox", "" );
EXCOMBOBOXLib::IComboBoxPtr spComboBox = editor.GetUserEditorObject();
if ( spComboBox != NULL )
{
   spComboBox->BeginUpdate();
   spComboBox->LabelHeight = m_xmlgrid.GetNodeHeight() - 3;
   spComboBox->LinesAtRoot = EXCOMBOBOXLib::exLinesAtRoot;
   spComboBox->put_HeightList( vtMissing, 256 );
   spComboBox->put_WidthList( vtMissing, 256 );
   spComboBox->IntegralHeight = true;
   spComboBox->Columns->Add("Name");
   spComboBox->Columns->Add("Value");
   spComboBox->ColumnAutoResize = true;
   EXCOMBOBOXLib::IItemsPtr spItems = spComboBox->Items;
   long h = spItems->AddItem(COleVariant( "Item 1" ));
   spItems->put_CellCaption(COleVariant(h),COleVariant((long)1), COleVariant( "Item 1.2" )
);
   long h1 = spItems->InsertItem(h, vtMissing, COleVariant( "SubItem 1") );
   spItems->put_CellCaption(COleVariant(h1),COleVariant((long)1), COleVariant( "SubItem
1.2" ) );
   h1 = spItems->InsertItem(h, vtMissing, COleVariant( "SubItem 2") );
   spItems->put_CellCaption(COleVariant(h1),COleVariant((long)1), COleVariant( "SubItem
```

```
2.2" ) );
   spItems->put_ExpandItem(h, true );
   spComboBox->EndUpdate();
}
```

The sample requires the #import <excombobox.dll> to include the ExComboBox's type library. The #import <excombobox.dll> creates EXCOMBOBOXLib namespace that includes all definitions for objects and types that the ExComboBox control exports.

The following VB.NET sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
With AxXMLGrid1.Editors
   With .Add("excombobox", EXMLGRIDLib.EditTypeEnum.UserEditorType)
      .UserEditor("Exontrol.ComboBox", "")
      With .UserEditorObject
         .BeginUpdate()
         .LabelHeight = AxXMLGrid1.NodeHeight - 3
         .LinesAtRoot = True
         .HeightList = 256
         .WidthList = 256
         .IntegralHeight = True
         .Columns.Add("Name")
         .Columns.Add("Value")
         .ColumnAutoResize = True
         With .Items
            Dim h, h1 As Integer
            h = .AddItem("Item 1")
            .CellCaption(h, 1) = "Item 1.2"
            h1 = .InsertItem(h, , "SubItem 1")
            .CellCaption(h1, 1) = "SubItem 1.2"
            h1 = .InsertItem(h, , "SubItem 2")
            .CellCaption(h1, 1) = "SubItem 2.2"
            .ExpandItem(h) = True
         End With
         .EndUpdate()
      End With
   End With
End With
```

The following C# sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
EXMLGRIDLib.Editor editor = axXMLGrid1.Editors.Add("excombobox",
EXMLGRIDLib.EditTypeEnum.UserEditorType);
editor.UserEditor("Exontrol.ComboBox", "");
EXCOMBOBOXLib.ComboBox comboBox = editor.UserEditorObject as
EXCOMBOBOXLib.ComboBox;
if ( comboBox != null )
{
   comboBox.BeginUpdate();
   comboBox.LabelHeight = axXMLGrid1.NodeHeight - 3;
   comboBox.LinesAtRoot = EXCOMBOBOXLib.LinesAtRootEnum.exLinesAtRoot ;
   comboBox.set_HeightList( null, 256 );
   comboBox.set_WidthList( null, 256 );
   comboBox.IntegralHeight = true;
   comboBox.Columns.Add("Name");
   comboBox.Columns.Add("Value");
   comboBox.ColumnAutoResize = true;
   EXCOMBOBOXLib.Items items = comboBox.Items;
   int h = items.AddItem("Item 1");
   items.set_CellCaption(h, 1, "Item 1.2" );
   int h1 = items.InsertItem(h, null, "SubItem 1");
   items.set_CellCaption(h1, 1,"SubItem 1.2");
   h1 = items.InsertItem(h, null, "SubItem 2");
   items.set_CellCaption(h1, 1,"SubItem 2.2");
   items.set_ExpandItem(h, true);
   comboBox.EndUpdate();
}
```

In C# your project needs a new reference to the Exontrol's ExComboBox control library. Use the Project\Add Reference\COM item to add new reference to a COM object. Once that you added a reference to the Exontrol's ExComboBox the EXCOMBOBOXLib namespace is created. The EXCOMBOBOXLib namespace contains definitions for all objects that ExComboBox control exports.

The following VFP sample adds an ActiveX editor, (Exontrol's ExComboBox):

```
With thisform.XMLGrid1.Editors
   With .Add("excombobox", 16) && UserEditorType
```

```
      .UserEditor("Exontrol.ComboBox", "")
      With .UserEditorObject
        .BeginUpdate
        .LabelHeight = thisform.XMLGrid1.NodeHeight - 3
        .LinesAtRoot = -1
        .HeightList(0) = 256
        .WidthList(0) =256
        .IntegralHeight = .t.
        .Columns.Add ("Name")
        .Columns.Add ("Value")
        .ColumnAutoResize = .t.
        With .Items
          .DefaultItem = .AddItem("Item 1")
          h = .DefaultItem
          .CellCaption(0, 1) = "Item 1.2"
          .DefaultItem = .InsertItem(h, , "SubItem 1")
          .CellCaption(0, 1) = "SubItem 1.2"
          .DefaultItem = .InsertItem(h, , "SubItem 2")
          .CellCaption(0, 1) = "SubItem 2.2"
          .DefaultItem = h
          .ExpandItem(0) = .t.
        EndWith
        .EndUpdate
      EndWith
    EndWith
EndWith
```

Editors object

The Editors collection holds a collection of Editor objects. Use the Editors property to access the control's editors collection. Use the Editor property to assign an editor to a node. The Editors collection supports the following properties and methods:

Name	Description
Add	Adds a child editor and returns a reference to the newly created object.
Clear	Removes all objects in the collection.
Count	Returns the number of objects in a collection.
Item	Returns a specific editor of the Editors collection.
ItemByKey	Returns an editor giving its key.
Remove	Removes a specific member from the Editors collection.
RemoveByKey	Removes an editor giving its key.

method Editors.Add (Key as Variant, Type as EditTypeEnum)

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

Type	Description
Key as Variant	A string or long expression that indicates the key of the editor being added.
Type as EditTypeEnum	An EditTypeEnum expression that indicates the type of the editor being added.

Return	Description
Editor	An Editor object being created.

Use the Add method to add new type of editors to the control. Use the Editor property to assign an editor to a node. Use the EditType property to change the type of editor. Use the Option property to define options for a specific type of editor. Use the AddItem method to add new items to a drop down editor. Use the AddButton method to insert buttons to the editor. The control fires the ButtonClick event when user presses a button inside an editor. Use the Change event to notify your application that user alters the node's value or caption.

Use the AutoEdit property to specify whether the control starts editing the focused node as soon as user moves the focused node. Use the Edit method to programmatically edit a node, if the AutoEdit property is False.

If a node has an editor assigned the node's editor is applied to the:

- Name property, if the node contains child node.
- Value property, if the node contains no child node.

The following sample adds a drop down editor and a float edit box:

```
With XMLGrid1
  .BeginUpdate
    .AutoEdit = True
    With .Editors.Add("Float", EditType)
      .Numeric = exFloat
    End With
    With .Editors.Add("DropDown", DropDownListType)
      .AddItem 1, "Yes"
      .AddItem 2, "No"
    End With
    With .Nodes
```

```
        With .Add("Root").Nodes
            With .Add("Child 1", "1.2")
                .Editor = "Float"
            End With
            With .Add("Child 2", "1")
                .Editor = "DropDown"
            End With
        End With
    End With
  .EndUpdate
End With
```

Use the AddNode event to apply a specific editor to all nodes in the control at adding time.

method Editors.Clear ()

Removes all objects in the collection.

Type	Description

Use the Clear method to remove the editors collection. Use the Remove method to remove a specific editor. Use the Editor property to assign am editor to a node.

property Editors.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	A long expression that indicates the number of elements in the collection.

The Count property specifies the number of editors in the collection. Use the Item, ItemByKey properties to access an editor by its index or key.

The following sample displays the list of control's editors collection:

```
Dim e As EXMLGRIDLibCtl.Editor
For Each e In XMLGrid1.Editors
   Debug.Print e.Key
Next
```

property Editors.Item (Index as Variant) as Editor

Returns a specific editor of the Editors collection.

Type	Description
Index as Variant	A long expression that indicates the index of the editor being accessed.
Editor	An Editor object being requested.

Use the Item, ItemByKey properties to access an editor by its index or key. The Count property specifies the number of editors in the collection.

The following sample displays the list of control's editors collection:

```
Dim e As EXMLGRIDLibCtl.Editor
For Each e In XMLGrid1.Editors
   Debug.Print e.Key
```

property Editors.ItemByKey (Key as Variant) as Editor

Returns an editor giving its key.

Type	Description
Key as Variant	A long or string expression that indicates the key of the editor being accessed.
Editor	An Editor object being requested.

Use the Item, ItemByKey properties to access an editor by its index or key. The Count property specifies the number of editors in the collection.

The following sample displays the list of control's editors collection:

```
Dim e As EXMLGRIDLibCtl.Editor
For Each e In XMLGrid1.Editors
   Debug.Print e.Key
```

method Editors.Remove (Index as Variant)

Removes a specific member from the Editors collection.

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

Use the Remove method to remove a specific editor giving its index. Use the RemoveByKey method to remove an editor giving its key. Use the Clear method to remove the editors collection. Use the Editor property to assign am editor to a node.

method Editors.RemoveByKey (Key as Variant)

Removes an editor giving its key.

Type	Description
Key as Variant	A string or long expression that indicates the key of the editor.

Use the RemoveByKey method to remove an editor giving its key. Use the Remove method to remove a specific editor giving its index. Use the Clear method to remove the editors collection. Use the Editor property to assign am editor to a node.

ExDataObject object

Defines the object that contains OLE drag and drop information.

Name	Description
Clear	Deletes the contents of the ExDataObject object.
Files	Returns an ExDataObjectFiles collection, which in turn contains a list of all filenames used by an ExDataObject object.
GetData	Returns data from an ExDataObject object in the form of a variant.
GetFormat	Returns a value indicating whether an item in the ExDataObject object matches a specified format.
SetData	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.

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
ExDataObjectFiles	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.

method ExDataObject.GetData (Format as Integer)

Returns data from a DataObject object in the form of a variant.

Type	Description
Format as Integer	An exClipboardFormatEnum 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 exClipboardFormatEnum 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 exClipboardFormatEnum 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.

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
Add	Adds a filename to the Files collection
Clear	Removes all file names in the collection.
Count	Returns the number of file names in the collection.
Item	Returns an specific file name.
Remove	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.

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.

Node object

The Node object holds information about control's node. Use the Nodes property to access the control's Nodes collection. Use the Add method to add a new node to the control. Use the Editors property to access the control's editors. The Node object supports the following properties and methods:

Name	Description
BackColor	Specifies the node's background color.
BackColorChild	Specifies the default background color for child nodes.
BackColorValue	Specifies the background color for the node's value.
BackgroundExt	Indicates additional colors, text, images that can be displayed on the node's background using the EBN string format.
BackgroundExtValue	Specifies at runtime, the value of the giving property for specified part of the background extension.
ClearBackColor	Clears the node's background color.
ClearBackColorChild	Clears the default background color for child nodes.
ClearBackColorValue	Clears the background of the node's value.
ClearForeColor	Clears the node's foreground color.
ClearForeColorChild	Clears the default foreground color for the child nodes.
ClearForeColorValue	Clears the foreground color for the node's value.
CollapseAll	Collapses all the child nodes.
Editor	Specifies a value that indicates the key of the node's editor.
Enabled	Specifies whether the node is enabled or disabled.
ExpandAll	Expands all the child nodes.
Expanded	Specifies whether a node is expanded or collapsed.
FirstNode	Gets the first child tree node in the tree node collection.
ForeColor	Specifies the node's background color.
ForeColorChild	Specifies the default foreground color for child nodes.
ForeColorValue	Specifies the foreground color for the node's value.
HasChilds	Specifies whether the node contains child nodes.
ID	Retrieves the node's unique identifier.
	Retrieves or sets a value that indicates the index of icon to

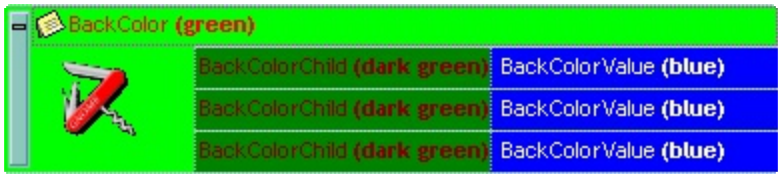
Image	display in the node.
Index	Retrieves the index of the node within the collection.
IsChildOf	Specifies whether a node is child of another node.
Key	Retrieves the node's key.
LastNode	Gets the last child tree node.
Level	Specifies the node's level.
Name	Specifies the caption of the node.
NextNode	Gets the next sibling tree node.
NextVisibleNode	Gets the next visible tree node.
Nodes	Gets the collection of Node objects assigned to the current node.
Parent	Retrieves the parent node.
Picture	Assign a picture to a node.
Position	Specifies the position of the node within the nodes collection.
PrevNode	Gets the previous sibling tree node.
PrevVisibleNode	Gets the previous visible tree node.
Selected	Specifies whether the node is selected.
ToolTip	Specifies the node's tooltip.
ToolTipTitle	Specifies the node's title for its tooltip.
UserData	Associates an extra data to the node.
Value	Specifies the value of the node.
Visible	Specifies whether a node is visible or hidden.

property Node.BackColor as Color

Specifies the node's background color.

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

Use the BackColor property to specify the node's background color. While the node's BackColor property is not specified the control uses the BackColor property to paint the node's background. Use the BackColorChild property to specify the background color for child nodes. Use the ClearBackColor method to clear the node's background color. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the BackColorValue property to specify the background color of the node's value. Use the SelBackColor, SelForeColor, SelBackColorChild and SelForeColorChild properties to customize the colors for selected nodes. Use the <bgcolor> built-in HTML format to specify a background color for parts of the node's value or name. Use the ForeColor property to specify the node's foreground color. Use the BackColorValue property to specify the node's value background color.



The following sample changes the node's background color:

```
Dim s As String
s =
"gBHJJGHA5MIwAEIe4AAAFhwFBwOCERDYXC4bEAgEopFIwiwwjgwGQyHcRHcZHcjHcrHZE

s = s +
"6jMbwHiGXQSHiAJSicDYYjYYROACUYyCaiIbBSOh4giQJCAUXY8ogGBhAMBxNBKKxECgA

s = s +
"YVECHAiFUTAmAgi+DyIUcAwwICKGaMAIYHQ3BkDiMQDYWRAABEMBcHQcAwBBAuDcBg



With XMLGrid1
   .BeginUpdate
      .LevelWidth(1) = 148
      .LevelWidth(2) = 128
```

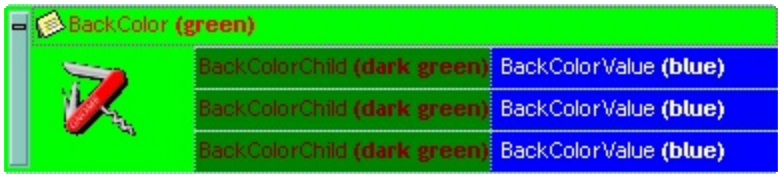
```
    With .Nodes
      With .Add("BackColor <b>(green)</b>")
        .Picture = s
        .Image = 1
        .BackColor = vbGreen
        .ForeColor = vbRed
        .BackColorChild = RGB(0, 128, 0)
        .ForeColorChild = RGB(128, 0, 0)

        With .Nodes
          With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 1)
            .BackColorValue = vbBlue
            .ForeColorValue = vbWhite
          End With
          With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 2)
            .BackColorValue = vbBlue
            .ForeColorValue = vbWhite
          End With
          With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 3)
            .BackColorValue = vbBlue
            .ForeColorValue = vbWhite
          End With
        End With

        .Expanded = True
      End With
    End With
  .EndUpdate
End With
```

property Node.BackColorChild as Color

Specifies the default background color for child nodes.

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

Use the BackColorChild property to specify the background color for child nodes. Use the BackColor property to specify the node's background color. While the node's BackColorChild property is not specified the control uses the BackColor property to paint the node's background. Use the ClearBackColorChild method to clear the child node's background color. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the BackColorValue property to specify the background color of the node's value. Use the SelBackColor, SelForeColor, SelBackColorChild and SelForeColorChild properties to customize the colors for selected nodes. Use the <bgcolor> built-in HTML format to specify a background color for parts of the node's value or name. Use the ForeColorChild property to specify the child node's foreground color.



The following sample changes the node's background color:

```
Dim s As String
s =
"gBHJJGHA5MIwAEIe4AAAFhwFBwOCERDYXC4bEAgEopFIwiwwjgwGQyHcRHcZHcjHcrHZE

s = s +
"6jMbwHiGXQSHiAJSicDYYjYYROACUYyCaiIbBSOh4giQJCAUXY8ogGBhAMBxNBKKxECgA

s = s +
"YVECHAiFUTAmAgi+DyIUcAwwICKGaMAIYHQ3BkDiMQDYWRAABEMBcHQcAwBBAuDcBg



With XMLGrid1
   .BeginUpdate
      .LevelWidth(1) = 148
      .LevelWidth(2) = 128
```

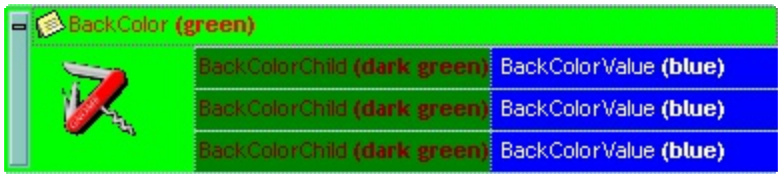
```
    With .Nodes
      With .Add("BackColor <b>(green)</b>")
        .Picture = s
        .Image = 1
        .BackColor = vbGreen
        .ForeColor = vbRed
        .BackColorChild = RGB(0, 128, 0)
        .ForeColorChild = RGB(128, 0, 0)

        With .Nodes
          With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 1)
            .BackColorValue = vbBlue
            .ForeColorValue = vbWhite
          End With
          With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 2)
            .BackColorValue = vbBlue
            .ForeColorValue = vbWhite
          End With
          With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 3)
            .BackColorValue = vbBlue
            .ForeColorValue = vbWhite
          End With
        End With

        .Expanded = True
      End With
    End With
  .EndUpdate
End With
```

property Node.BackColorValue as Color

Specifies the background color for the node's value.

Type	Description
Color	A color expression that indicates the background color for the node's value.

Use the BackColorValue property to specify the node's value background color. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the Value property to change the node's value. Use the Name property to assign a new name to a node. Use the BackColor property to specify the node's background color. Use the BackColorChild property to specify the background color for child nodes. Use the ForeColorValue property to specify the node's value foreground color. Use the ClearBackColorValue method to clear the node's value background color.



property Node.BackgroundExt(State as BackgroundExtStateEnum) as String

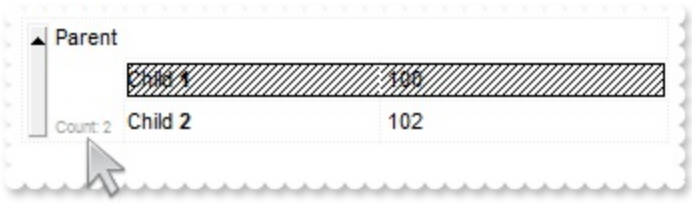
Indicates additional colors, text, images that can be displayed on the object's background using the EBN string format.

Type	Description
State as BackgroundExtStateEnum	A BackgroundExtStateEnum expression that indicates where the background extension is applied.
String	A String expression (**"EBN String Format"**) that defines the layout of the UI to be applied on the object's background. The syntax of EBN String Format in BNF notation is shown bellow. *You can use the EBN's Builder of eXButton/COM control to define visually the EBN String Format.*

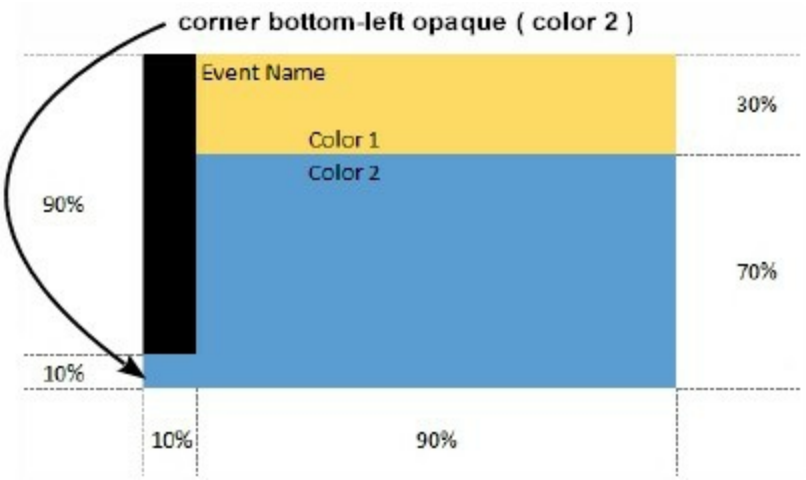
By default, the BackgroundExt property is empty. Using the BackgroundExt property you have unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the object's background. *For instance, let's say you need to display **more** colors on the object's background, or just 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 BackgroundExtValue property to change at runtime any UI property for any part that composes the EBN String Format. The BackgroundExt property is applied right after setting the object's backcolor, and before drawing the default object's captions, icons or pictures.

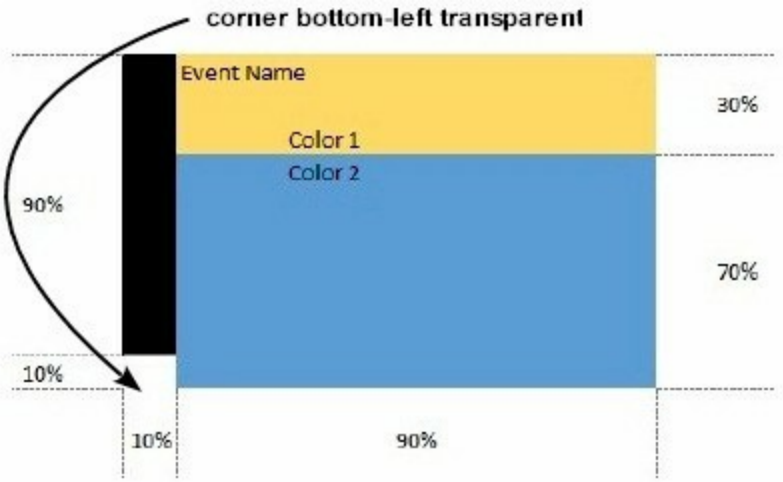
The following screen shot shows how you can extend the node as follows:

- displays the picture to a different place
- assign more HTML captions to the node
- different type of borders/frames
- and so on.



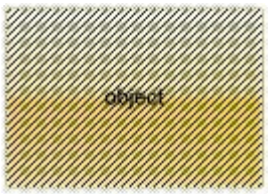
Complex samples:



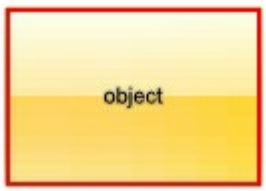


Easy samples:

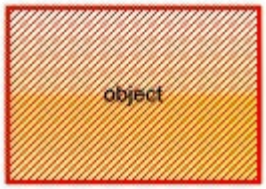
- "[pattern=6]", shows the BDiagonal pattern on the object's background.



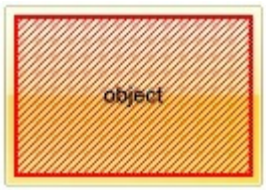
- "[frame=RGB(255,0,0),framethick]", draws a red thick-border around the object.



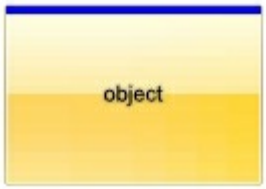
- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a red thick-border around the object, with a patter inside.



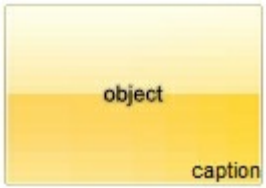
- "[[patterncolor=RGB(255,0,0)](none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(25 draws a red thick-border around the object, with a patter inside, with a 4-pixels wide padding:



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

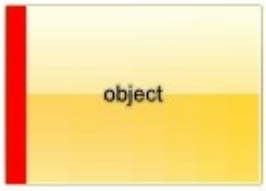


- "[text=`caption`,align=0x22]", shows the caption string aligned to the bottom-right side of the object's background.

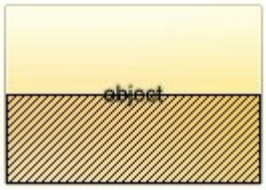


- "[text=`<img>flag</img>`,align=0x11]" shows the flag picture and the sweden string aligned to the bottom side of the object.

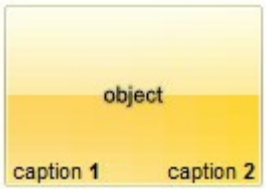
- "left[10,back=RGB(255,0,0)]", draws a red line on the left side of the object's background, of 10-pixels wide.



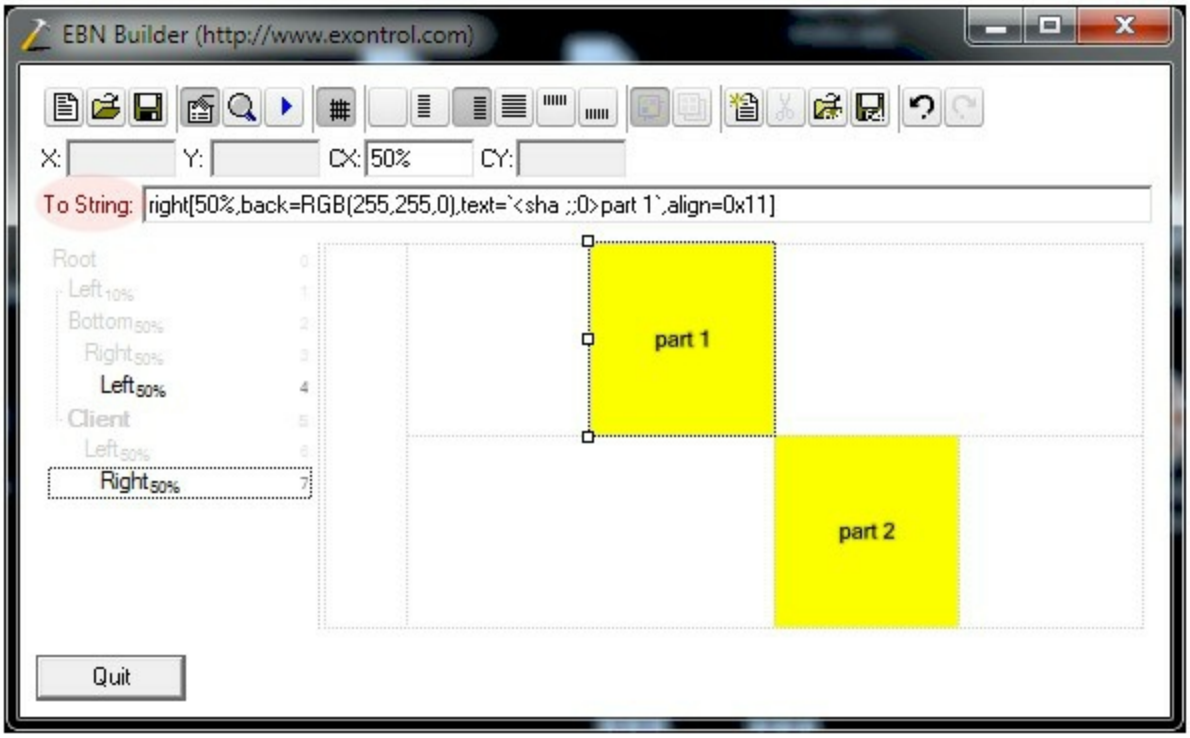
- "bottom[50%,pattern=6,frame]", shows the BDiagonal pattern with a border arround on the lower-half part of the object's background.



- "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 WYSWYG Builder helps you to generate or view the EBN String Format, in the **To String** field as shown in the following screen shot:*



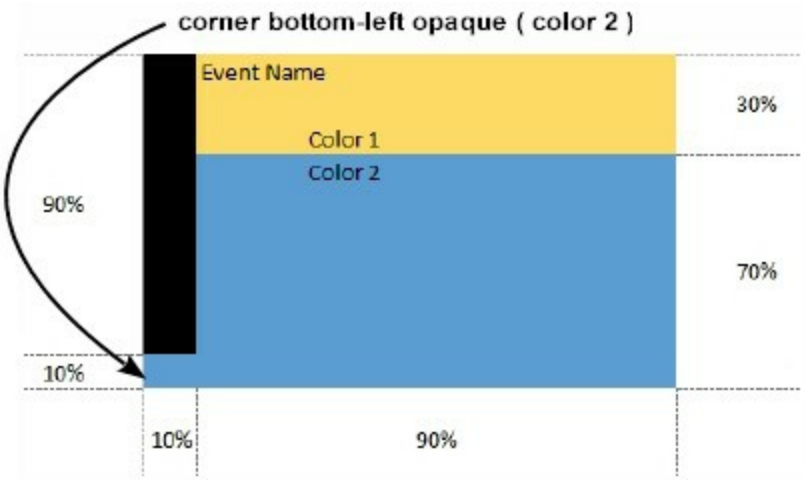
The **To String** field of the EBN Builder defines the **EBN String Format** that can be used on BackgroundExt property.

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

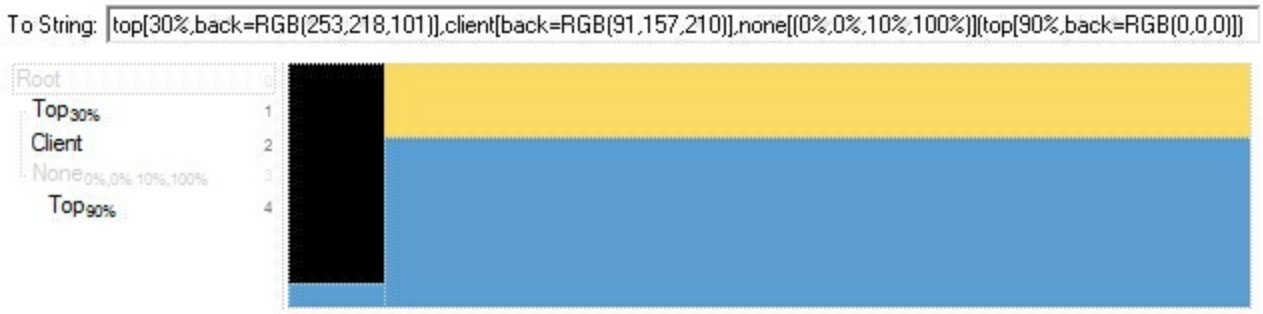
```
<EBN> ::= <elements> | <root> "(" [<elements>] ")"
<elements> ::= <element> [ "," <elements> ]
<root> ::= "root" [ <attributes> ] | [ <attributes> ]
<element> ::= <anchor> [ <attributes> ] [ "(" [<elements>] ")" ]
<anchor> ::= "none" | "left" | "right" | "client" | "top" | "bottom"
<attributes> ::= "[" [<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>
<backcolor> ::= "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.

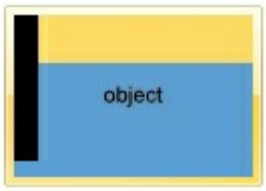
Now, lets say we have the following request to layout the colors on the objects:



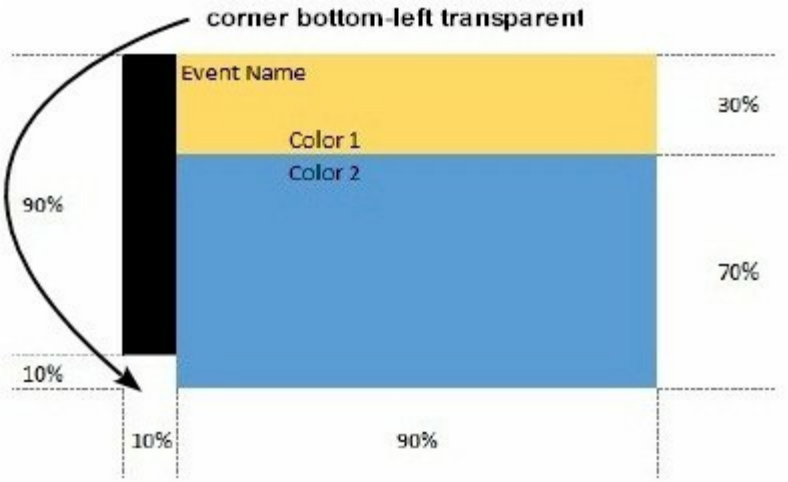
We define the BackgroundExt property such as "top[30%,back=RGB(253,218,101)],client[back=RGB(91,157,210)],none[(0%,0%,10%,100%)](top[90%,back=RGB(0,0,0)])", and it looks as:



so, if we apply to our object we got:

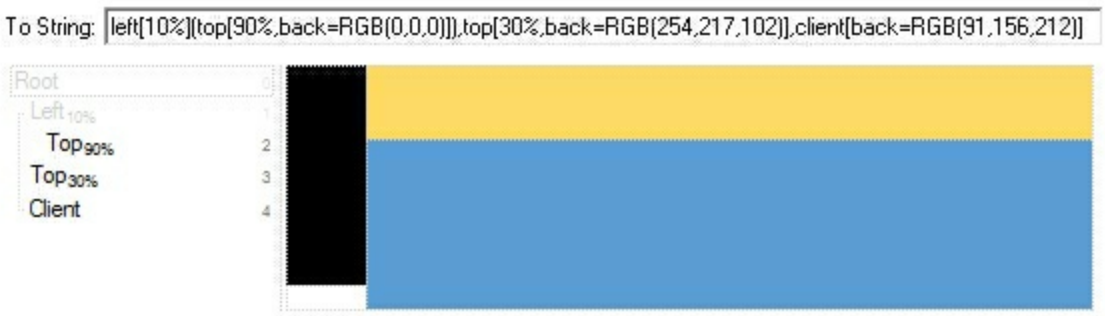


Now, lets say we have the following request to layout the colors on the objects:

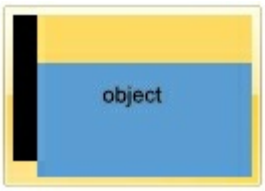


We define BackgroundExt property such as "left[10%]

(top[90%,back=RGB(0,0,0)]),top[30%,back=RGB(254,217,102)],client[back=RGB(91,156,2
and it looks as:

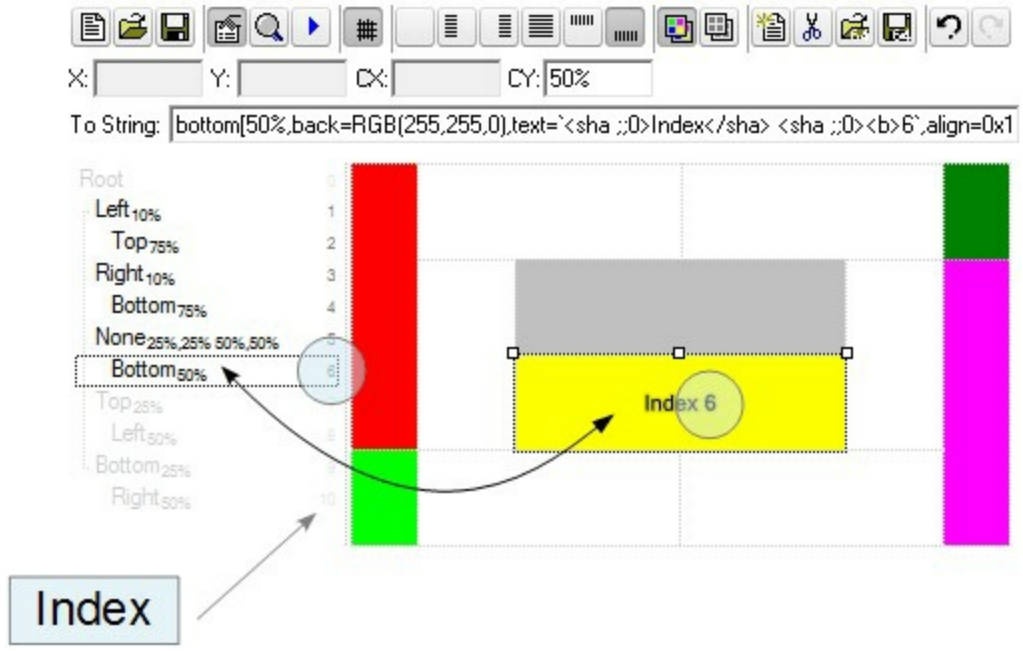


so, if we apply to our object we got:



property Node.BackgroundExtValue(State as BackgroundExtStateEnum, 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
State as BackgroundExtStateEnum	A BackgroundExtStateEnum expression indicates the state of the node where the value is applied to node's background extenstion.
Index as IndexExtEnum	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:  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 BackgroundExtPropertyEnum	A BackgroundExtPropertyEnum 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 BackgroundExt 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.*

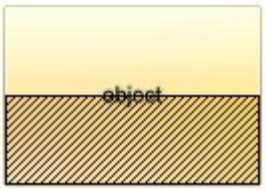
You can access/define/change the following UI properties of the element:

- **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

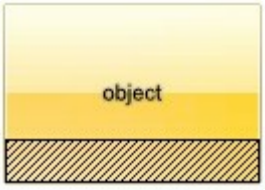
property set the Frame flag for exPatternExt property. *(Color expression)*
- **exFrameThickExt**(11), Specifies that a thick-frame is shown around the object. This property set the FrameThick flag for exPatternExt property. *(Boolean expression)*
- **exUserDataExt**(12), Specifies an extra-data associated 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%"
```

method Node.ClearBackColor ()

Clears the node's background color.

Type	Description

Use the ClearBackColor method to clear the node's background color. Use the BackColor property to specify the node's background color. Use the ClearBackColorChild method to clear the background color of the child nodes.

The following sample changes the node's background color while cursor hovers the control:

```
Dim nOld As EXMLGRIDLibCtl.Node
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
      h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
      If Not nOld Is n Then
         If Not nOld Is Nothing Then
            nOld.ClearBackColor
            nOld.ClearForeColor
         End If
         If Not h = 0 Then
            n.BackColor = vbGreen
            n.ForeColor = vbBlue
         End If
      End If
      Set nOld = n
   End With
End Sub
```

method Node.ClearBackColorChild ()

Clears the default background color for child nodes.

Type	Description

Use the ClearBackColorChild method to clear the background color of the child nodes. Use the BackColorChild property to specify the child node's background color. Use the ClearBackColor method to clear the node's background color.

The following sample changes the node's background color while cursor hovers the control:

```
Dim nOld As EXMLGRIDLibCtl.Node
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
      h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
      If Not nOld Is n Then
         If Not nOld Is Nothing Then
            nOld.ClearBackColor
            nOld.ClearForeColor
         End If
         If Not h = 0 Then
            n.BackColor = vbGreen
            n.ForeColor = vbBlue
         End If
      End If
      Set nOld = n
   End With
End Sub
```

method Node.ClearBackColorValue ()

Clears the background of the node's value.

Type	Description

Use the ClearBackColorValue method to clear the node's value background color. Use the BackColorValue property to specify the node's value background color. Use the BackColor property to specify the node's background color. Use the ForeColor property to specify the node's foreground color. Use the ForeColorValue property to specify the node's value foreground color.

The following sample changes the node's value background color while cursor hovers the control:

```
Dim nOld As EXMLGRIDLibCtl.Node
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
      h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
      If Not nOld Is n Then
         If Not nOld Is Nothing Then
            nOld.ClearBackColorValue
            nOld.ClearForeColorValue
         End If
         If Not h = 0 Then
            n.BackColorValue = vbGreen
            n.ForeColorValue = vbBlue
         End If
      End If
      Set nOld = n
   End With
End Sub
```

method Node.ClearForeColor ()

Clears the node's foreground color.

Type	Description

Use the ClearForeColor method to clear the node's foreground color. Use the ForeColor property to specify the node's foreground color. Use the ClearForeColorChild method to clear the foreground color of the child nodes.

The following sample changes the node's background color while cursor hovers the control:

```
Dim nOld As EXMLGRIDLibCtl.Node
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
      h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
      If Not nOld Is n Then
         If Not nOld Is Nothing Then
            nOld.ClearBackColor
            nOld.ClearForeColor
         End If
         If Not h = 0 Then
            n.BackColor = vbGreen
            n.ForeColor = vbBlue
         End If
      End If
      Set nOld = n
   End With
End Sub
```

method Node.ClearForeColorChild ()

Clears the default foreground color for the child nodes.

Type	Description

Use the ClearForeColorChild method to clear the foreground color of the child nodes. Use the ForeColorChild property to specify the child node's foreground color. Use the ClearForeColor method to clear the node's foreground color.

The following sample changes the node's Foreground color while cursor hovers the control:

```
Dim nOld As EXMLGRIDLibCtl.Node
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
      h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
      If Not nOld Is n Then
         If Not nOld Is Nothing Then
            nOld.ClearBackColor
            nOld.ClearForeColor
         End If
         If Not h = 0 Then
            n.BackColor = vbGreen
            n.ForeColor = vbBlue
         End If
      End If
      Set nOld = n
   End With
End Sub
```

method Node.ClearForeColorValue ()

Clears the foreground color for the node's value.

Type	Description

Use the ClearForeColorValue method to clear the node's value foreground color. Use the ForeColorValue property to specify the node's value foreground color. Use the BackColor property to specify the node's background color. Use the ForeColor property to specify the node's foreground color. Use the ForeColorValue property to specify the node's value foreground color.

The following sample changes the node's value Foreground color while cursor hovers the control:

```
Dim nOld As EXMLGRIDLibCtl.Node
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
      h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
      If Not nOld Is n Then
         If Not nOld Is Nothing Then
            nOld.ClearForeColorValue
            nOld.ClearForeColorValue
         End If
         If Not h = 0 Then
            n.ForeColorValue = vbGreen
            n.ForeColorValue = vbBlue
         End If
      End If
      Set nOld = n
   End With
End Sub
```

method Node.CollapseAll ()

Collapses all the child nodes.

Type	Description

Use the CollapseAll method to collapse all child nodes. Use the ExpandAll method to expand all child nodes. Use the CollapseAll method to collapse all nodes in the control. Use the ExpandAll method to expand all nodes in the control. Use the Expanded property to expand or collapse a node.

property Node.Editor as Variant

Specifies a value that indicates the key of the node's editor.

Type	Description
Variant	A EditTypeEnum, string, numeric expression that indicates the key of the editor being assigned to a node. The control automatically adds a new editor of EditTypeEnum type, if no editor with specified key is found. Ability to specify the node's editor without calling the Editors.Add before, by specify the Node.Editor property to a EditTypeEnum value. For instance, Nodes.Add("Date", Date).Editor = EXMLGRIDLibCtl.EditTypeEnum.DateType, adds a node with a DateType editor.

The Editor property indicates the key of the editor being assigned to the node. If the Editor property indicates a key of an editor that doesn't exist in the Editors collection, it has no effect. Use the Editors property to access the control's Editors collection. Use the AutoEdit property to specify whether the control starts editing the focused node as soon as user moves the focused node.

Use the Add method to add new type of editors to the control's editors collection. A node displays only the node's name if the node contains child nodes, else it displays the node's name and the node's value. Use the Value property to assign a value to a node. If a node has an editor assigned, it changes the node's name if the node contains child nodes, else it changes the node's value if the node has no child nodes. Use the Add method to add new nodes to the control.

The control fires the Change event when the user changes the node's value if the node has no child nodes, or the node's name if the node has child nodes.

If a node has an editor assigned the node's editor is applied to the:

- Name property, if the node contains child node.
- Value property, if the node contains no child node.

The following sample adds a node that has a check list editor associated:

```
With XMLGrid1
  .BeginUpdate
    With .Editors
      With .Add("Check")
        .EditType = CheckListType
```

```
            .AddItem 1, "<b>1</b> One"
            .AddItem 2, "<b>2</b> Two"
            .AddItem 4, "<b>4</b> Four"
        End With
    End With
    With .Nodes
        With .Add("CheckList <b>type</b>", 3)
            .Editor = "Check"
        End With
    End With
  .EndUpdate
End With
```

property Node.Enabled as Boolean

Specifies whether the node is enabled or disabled.

Type	Description
Boolean	A boolean expression that indicates whether a node is enabled or disabled.

Use the Enabled property to disable a node. Use the Enabled property to disable the control. Use the Visible property to hide a node. A disabled node cann't be edited. Use the Editor property to remove the node's editor.

method Node.ExpandAll ()

Expands all the child nodes.

Type	Description

Use the ExpandAll method to expand all child nodes. Use the CollapseAll method to collapse all child nodes. Use the CollapseAll method to collapse all nodes in the control. Use the ExpandAll method to expand all nodes in the control. Use the Expanded property to expand or collapse a node.

property Node.Expanded as Boolean

Specifies whether a node is expanded or collapsed.

Type	Description
Boolean	A boolean expression that indicates whether a node is expanded or collapsed.

Use the Expanded property to expand or collapse a node. The control fires the BeforeExpandNode event before expanding or collapsing a node. The control fires the AfterExpandNode event to notify your application that a node is expanded or collapsed. Use the ExpandAll method to expand all child nodes. Use the CollapseAll method to collapse all child nodes. Use the ExpandAll method to expand all nodes in the control. Use the CollapseAll method to collapse all nodes in the control. Use the HasChilds property to specify whether the node displays the +/- sign to build your virtual tree. Use the ExpandOnDblClk property to let users expand or collapse nodes when double clicking a node. Use the ExpandOnKeys property to allow users expand or collapse the nodes using the keyboard. Use the ExpandButtons property to assign a different appearance for expanding/collapsing buttons. The ExpandOnSearch property specifies whether the control expands nodes when incremental searching is on (AutoSearch property is different than 0) and user types characters when the control has the focus.

property Node.FirstNode as Node

Gets the first child tree node in the tree node collection.

Type	Description
Node	A Node object that indicates the first child node.

Use the FirstNode property to get the first child node. Use the NextNode property to get the next sibling node. Use the PrevNode property to get the previous sibling node. Use the Visible property to hide a node. Use the LastNode property to get the last child node. Use the NextVisibleNode property to get the next visible node. Use the PrevVisibleNode property to get the previous visible node. Use the FirstVisibleNode property to get the first visible node in the control's client area.

The following sample displays recursively all child nodes:

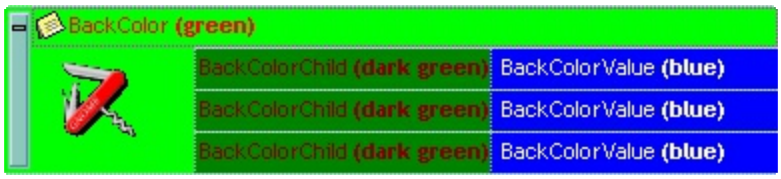
```
Private Sub scanRec(ByVal x As EXMLGRIDLibCtl.XMLGrid, ByVal n As
EXMLGRIDLibCtl.Node)
   Dim c As EXMLGRIDLibCtl.Node
   Set c = n.FirstNode
   While Not c Is Nothing
      Debug.Print c.Name
      scanRec x, c
      Set c = c.NextNode
   Wend
End Sub
```

property Node.ForeColor as Color

Specifies the node's Foreground color.

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

Use the ForeColor property to specify the node's foreground color. While the node's ForeColor property is not specified the control uses the ForeColor property to paint the node's foreground. Use the ForeColorChild property to specify the foreground color for child nodes. Use the ClearForeColor method to clear the node's foreground color. Use the ForeColorValue property to specify the Foreground color of the node's value. Use the SelForeColor, SelForeColor, SelForeColorChild and SelForeColorChild properties to customize the colors for selected nodes. Use the <fgcolor> built-in HTML format to specify a foreground color for parts of the node's value or name. Use the BackColor property to specify the node's background color.



The following sample changes the node's Foreground color:

```
Dim s As String
s =
"gBHJJGHA5MIwAEIe4AAAFhwFBwOCERDYXC4bEAgEopFIwiwwjgwGQyHcRHcZHcjHcrHZE

s = s +
"6jMbwHiGXQSHiAJSicDYYjYYROACUYyCaiIbBSOh4giQJCAUXY8ogGBhAMBxNBKKxECgA

s = s +
"YVECHAiFUTAmAgi+DyIUcAwwICKGaMAIYHQ3BkDiMQDYWRAABEMBcHQcAwBBAuDcBg



With XMLGrid1
   .BeginUpdate
      .LevelWidth(1) = 148
      .LevelWidth(2) = 128
      With .Nodes
```

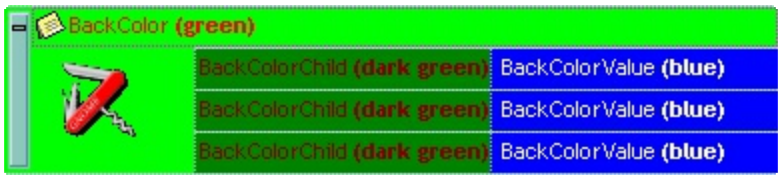
```
        With .Add("BackColor <b>(green)</b>")
            .Picture = s
            .Image = 1
            .BackColor = vbGreen
            .ForeColor = vbRed
            .BackColorChild = RGB(0, 128, 0)
            .ForeColorChild = RGB(128, 0, 0)

            With .Nodes
                With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 1)
                    .BackColorValue = vbBlue
                    .ForeColorValue = vbWhite
                End With
                With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 2)
                    .BackColorValue = vbBlue
                    .ForeColorValue = vbWhite
                End With
                With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 3)
                    .BackColorValue = vbBlue
                    .ForeColorValue = vbWhite
                End With
            End With

            .Expanded = True
        End With
    End With
  .EndUpdate
End With
```

property Node.ForeColorChild as Color

Specifies the default foreground color for child nodes.

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

Use the ForeColorChild property to specify the foreground color for child nodes. Use the ForeColor property to specify the node's foreground color. While the node's ForeColorChild property is not specified the control uses the ForeColor property to paint the node's Foreground. Use the ClearForeColorChild method to clear the child node's foreground color. Use the ForeColorValue property to specify the foreground color of the node's value. Use the SelForeColor, SelForeColor, SelForeColorChild and SelForeColorChild properties to customize the colors for selected nodes. Use the <fgcolor> built-in HTML format to specify a foreground color for parts of the node's value or name. Use the BackColorChild property to specify the child node's background color.



The following sample changes the node's background color:

```
Dim s As String
s =
"gBHJJGHA5MIwAEIe4AAAFhwFBwOCERDYXC4bEAgEopFIwiwwjgwGQyHcRHcZHcjHcrHZE

s = s +
"6jMbwHiGXQSHiAJSicDYYjYYROACUYyCaiIbBSOh4giQJCAUXY8ogGBhAMBxNBKKxECgA

s = s +
"YVECHAiFUTAmAgi+DyIUcAwwICKGaMAIYHQ3BkDiMQDYWRAABEMBcHQcAwBBAuDcBg



With XMLGrid1
   .BeginUpdate
      .LevelWidth(1) = 148
      .LevelWidth(2) = 128
      With .Nodes
```

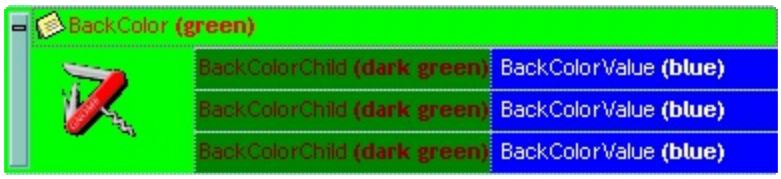
```
        With .Add("BackColor <b>(green)</b>")
            .Picture = s
            .Image = 1
            .BackColor = vbGreen
            .ForeColor = vbRed
            .BackColorChild = RGB(0, 128, 0)
            .ForeColorChild = RGB(128, 0, 0)

            With .Nodes
                With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 1)
                    .BackColorValue = vbBlue
                    .ForeColorValue = vbWhite
                End With
                With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 2)
                    .BackColorValue = vbBlue
                    .ForeColorValue = vbWhite
                End With
                With .Add("BackColorChild <b>(dark green)</b>", "BackColorValue <b>
(blue)</b>", 3)
                    .BackColorValue = vbBlue
                    .ForeColorValue = vbWhite
                End With
            End With

            .Expanded = True
        End With
    End With
  .EndUpdate
End With
```

property Node.ForeColorValue as Color

Specifies the foreground color for the node's value.

Type	Description
Color	A color expression that indicates the foreground color for the node's value.

Use the ForeColorValue property to specify the node's value foreground color. Use the Name property to assign a new name to a node. Use the Value property to change the node's value. Use the ForeColor property to specify the node's foreground color. Use the ForeColorChild property to specify the foreground color for child nodes. Use the BackColorValue property to specify the node's value background color. Use the ClearForeColorValue method to clear the node's value foreground color.



property Node.HasChilds as Boolean

Specifies whether the node contains child nodes.

Type	Description
Boolean	A boolean expression that indicates whether the node displays +/- signs even if the node contains no child nodes.

Use the HasChilds property to display expanding/collapsing buttons for a node to build your virtual tree. The property has no effect if the node contains already visible child nodes. Use the BeforeExpandNode event to notify your application that the user is about to expand or collapse a node. Use the Expanded property to expand or collapse a node. You can use the BeforeExpandNode event to cancel expanding specified nodes.

The following sample adds new child nodes to the node that's about to be expanded:

```
Private Sub XMLGrid1_BeforeExpandNode(ByVal Node As EXMLGRIDLibCtl.INode, Cancel
As Variant)
  If Not Node.Expanded Then
    With Node.Nodes
      With .Add("New Node")
        .HasChilds = True
      End With
    End With
  End If
End Sub
```

property Node.ID as Variant

Retrieves the node's unique identifier.

Type	Description
Variant	A String expression that determines the unique node identifier.

By default, the ID property is generated by the control, to identify uniquely a node within the Nodes collection. The ItemByID property gets the node giving its identifier. For instance, the ID property looks as: "0.1.1". The RemoveByID method removes a node giving its unique identifier.

property Node.Image as Long

Retrieves or sets a value that indicates the index of icon to display in the node.

Type	Description
Long	A long value that indicates the index of the icon in Images collection. The Images collection is 1 based.

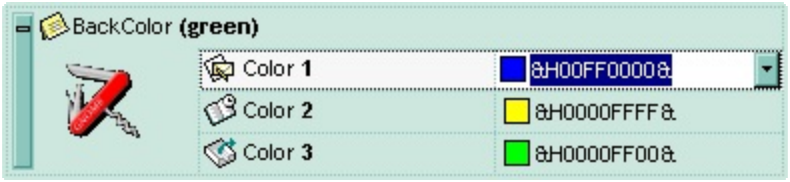
Use the Image property to assign an icon to a node. The node's icon is displayed on the left side of the node. The node's picture is displayed on the child level area. Use the Picture property to assign a picture to a node. Use the Images method to load icons to the control.

In case you are using the LoadXML method, the Image property of the Node indicates the type of XML node being added. The type of valid XML nodes are:

- **NODE_ELEMENT** (1) The node represents an element (its nodeTypeString property is "element"). An Element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection, and EntityReference. The Element node can be the child of the Document, DocumentFragment, EntityReference, and Element nodes.
- **NODE_ATTRIBUTE** (2) The node represents an attribute of an element (its nodeTypeString property is "attribute"). An Attribute node can have the following child node types: Text and EntityReference. The Attribute node does not appear as the child node of any other node type; it is not considered a child node of an Element.
- **NODE_TEXT** (3) The node represents the text content of a tag (its nodeTypeString property is "text"). A Text node cannot have any child nodes. The Text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.
- **NODE_CDATA_SECTION** (4) The node represents a CDATA section in the XML source (its nodeTypeString property is "cdatasection"). CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATASection node cannot have any child nodes. The CDATASection node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.
- **NODE_ENTITY_REFERENCE** (5) The node represents a reference to an entity in the XML document (its nodeTypeString property is "entityreference"). This applies to all entities, including character entity references. An EntityReference node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The EntityReference node can appear as the child of the Attribute, DocumentFragment, Element, and EntityReference nodes.
- **NODE_ENTITY** (6) The node represents an expanded entity (its nodeTypeString property is "entity"). An Entity node can have child nodes that represent the expanded entity (for example, Text and EntityReference nodes). The Entity node can appear as the child of the DocumentType node.

- **NODE_PROCESSING_INSTRUCTION** (7) The node represents a processing instruction from the XML document (its nodeTypeString property is "processinginstruction"). A ProcessingInstruction node cannot have any child nodes. The ProcessingInstruction node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
- **NODE_COMMENT** (8) The node represents a comment in the XML document (its nodeTypeString property is "comment"). A Comment node cannot have any child nodes. The Comment node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
- **NODE_DOCUMENT** (9) The node represents a document object, that as the root of the document tree, provides access to the entire XML document (its nodeTypeString property is "document"). It is created using the progID "Microsoft.XMLDOM" or through a data island using <XML> or <SCRIPT LANGUAGE=XML>. A Document node can have the following child node types: Element (maximum of one), ProcessingInstruction, Comment, and DocumentType. The Document node cannot appear as the child of any node types.
- **NODE_DOCUMENT_TYPE** (10) The node represents the document type declaration, indicated by the <!DOCTYPE> tag (its nodeTypeString property is "documenttype"). A DocumentType node can have the following child node types: Notation and Entity. The DocumentType node can appear as the child of the Document node.
- **NODE_DOCUMENT_FRAGMENT** (11) The node represents a document fragment (its nodeTypeString property is "documentfragment"). The DocumentFragment node associates a node or subtree with a document without actually being contained within the document. A DocumentFragment node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The DocumentFragment node cannot appear as the child of any node types.
- **NODE_NOTATION** (12) The node represents a notation in the document type declaration (its nodeTypeString property is "notation"). A Notation node cannot have any child nodes. The Notation node can appear as the child of the DocumentType node.

Use the Images method to add images to the control, so each type of element in your XML file, has a specific representation. The first icon in the Images collection indicates the NODE_ELEMENT type, the second icon in the Images collection indicates the NODE_ATTRIBUTE type, and so on.



property Node.Index as Long

Retrieves the index of the node within the collection.

Type	Description
Long	A long expression that indicates the index of the node in the Nodes collection.

The Index property specify the node's index in the control's nodes collection. Use the Item property to access a node by its index. Use the Key property to identify a node. Use the Add method to insert new nodes to the control's nodes collection.

property Node.IsChildOf (Parent as Node) as Boolean

Specifies whether a node is child of another node.

Type	Description
Parent as Node	A Node object that specifies the node's parent.
Boolean	A boolean expression that indicates whether the node is child of the Parent node.

Use the IsChildOf property to check whether a node is child of another node. Use the Parent property to get the node's parent.

property Node.Key as String

Retrieves the node's key.

Type	Description
String	A string expression that indicates the node's key.

Use the Key property to identify a node. Use the Item property to access a node by its key. The Index property specify the node's index in the control's nodes collection. Use the Add method to insert new nodes to the control's nodes collection.

property Node.LastNode as Node

Gets the last child tree node.

Type	Description
Node	A Node object that specifies the last child node.

Use the LastNode property to get the last child node. Use the FirstNode property to get the first child node. Use the NextNode property to get the next sibling node. Use the PrevNode property to get the previous sibling node. Use the Visible property to hide a node.

property Node.Level as Long

Specifies the node's level.

Type	Description
Long	A long expression that indicates the node's level.

The Level property indicates the node's level. A root node has the level 0. The child nodes of the root node has the level 1, and so on. The level of the child nodes is equal with the level of the parent node plus 1. Use the LevelWidth property to specify the level's width. Use the VisibleLevelCount property to specify the number of levels being displayed.

property Node.Name as String

Specifies the caption of the node.

Type	Description
String	A string expression that indicates the node's name (caption).

The Name property defines the node's caption. Use the Value property to assign a value to a node. Use the Editor property to assign an editor to a node. The control fires the Change event when the user changes the node's value if the node has no child nodes, or the node's name if the node has child nodes. Use the UserData property to assign an extra data to a node. Use the Add method to specify the node's name and value at adding time. If the node has an editor assigned, and it contains child nodes, the Name property indicates the value for the assigned editor. Use the BackColor property to specify the node's background color. Use the BackColorChild property to specify the background color for child nodes.

The Name property supports built-in HTML format like follows:

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

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

- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with $^{\text{subscript}}$
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra**

FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

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

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

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

property Node.NextNode as Node

Gets the next sibling tree node.

Type	Description
Node	A Node object that's the next sibling node.

Use the NextNode property to get the next sibling node. Use the FirstNode property to get the first child node. Use the PrevNode property to get the previous sibling node. Use the Visible property to hide a node. Use the NextVisibleNode property to get the next visible node. Use the PrevVisibleNode property to get the previous visible node. If there is no next tree node, the NextNode property returns a null reference (Nothing in Visual Basic).

The following sample displays recursively all child nodes:

```
Private Sub scanRec(ByVal x As EXMLGRIDLibCtl.XMLGrid, ByVal n As
EXMLGRIDLibCtl.Node)
   Dim c As EXMLGRIDLibCtl.Node
   Set c = n.FirstNode
   While Not c Is Nothing
      Debug.Print c.Name
      scanRec x, c
      Set c = c.NextNode
   Wend
End Sub
```

property Node.NextVisibleNode as Node

Gets the next visible tree node.

Type	Description
Node	A Node object that indicates the next visible node.

Use the NextVisibleNode property to get the next visible node. Use the FirstVisibleNode property to get the first visible node in the control's client area. Use the PrevVisibleNode property to get the previous visible node. Use the Visible property to hide a node. The NextVisibleNode can be a child, sibling, or a tree node from another branch. If there is no next tree node, the NextVisibleNode property returns a null reference (Nothing in Visual Basic).

The following sample displays the visible nodes in the control:

```
Private Sub vis(ByVal x As EXMLGRIDLibCtl.XMLGrid)
   Dim c As EXMLGRIDLibCtl.Node
   Set c = x.FirstVisibleNode
   While Not c Is Nothing
      Debug.Print c.Name
      Set c = c.NextVisibleNode
   Wend
End Sub
```

property Node.Nodes as Nodes

Gets the collection of Node objects assigned to the current node.

Type	Description
Nodes	A Nodes object that specifies the collection of child nodes.

Use the Nodes method to access the node's child nodes collection. Use the Add method to insert child nodes. Use the Editor property to assign an editor to a node. Use the Editors property to access the control's collection of editors. Use the Nodes property to access the control's nodes collection.

The following sample adds few nodes to the control's nodes collection.

```
Private Sub Form_Load()
  With XMLGrid1
    .BeginUpdate

    With .Nodes
      With .Add("Root").Nodes
        .Add "Child 1", "text1"
        .Add "Child 2", "text2"
      End With
    End With
    .EndUpdate
  End With
End Sub
```

property Node.Parent as Node

Retrieves the parent node.

Type	Description
Node	A Node object that specifies the node's parent.

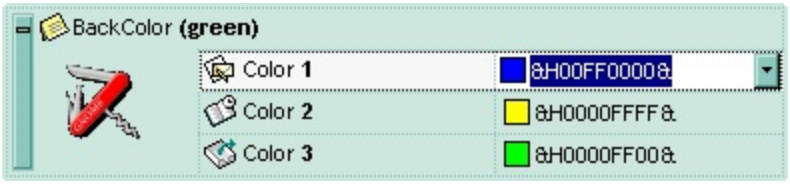
Use the Parent property to get the node's parent. Use the IsChildOf property to check whether a node is child of another node. Use the Nodes property to access the child node's collection. Use the Add method to add child nodes to a node. Use the Remove method to remove a node.

property Node.Picture as Variant

Assign a picture to a node.

Type	Description
Variant	• A Picture object that indicates the node's picture (A Picture object implements IPicture interface), • A String expression that specifies the path to picture's file to be displayed • A String expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format. • A String expression that specifies the key of HTMLPicture to be displayed.

Use the Picture property to assign a picture to a node. The node's icon is displayed on the left side of the node. The node's picture is displayed on the child level area, so the node should be expanded. The Expanded property specifies whether the node is expanded or collapsed. Use the Image property to assign an icon to a node. Use the Images method to load icons to the control. Use the Picture property to put a picture on the control's background. You can use the BackgroundExt property for unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the node's background



property Node.Position as Long

Specifies the position of the node within the nodes collection.

Type	Description
Long	A long expression that indicates the position of the node in the nodes collection.

Use the Position property to specify the node's position inside the nodes collection. Use the Visible property to hide a node. Use the Remove method to remove a node. Use the NodeHeight property to specify the height of the nodes. Use the FirstNode property to get the first node in the child nodes collection. Use the NextNode property to get the next sibling tree node. Use the NodeByPosition property to get a node giving its position. Use the FirstVisibleNode property to get the first visible node in the control's client area.

The following sample displays the list of visible nodes as they are displayed:

```
With XMLGrid1
   Dim n As EXMLGRIDLibCtl.Node, i As Long
   i = 0
   Set n = .NodeByPosition(i)
   While Not n Is Nothing
      Debug.Print n.Name
      i = i + 1
      Set n = .NodeByPosition(i)
   Wend
End With
```

property Node.PrevNode as Node

Gets the previous sibling tree node.

Type	Description
Node	A Node object that indicates the previous sibling node.

Use the PrevNode property to get the previous sibling node. Use the NextNode property to get the next sibling node. Use the FirstNode property to get the first child node. Use the Visible property to hide a node. Use the NextVisibleNode property to get the next visible node. Use the PrevVisibleNode property to get the previous visible node. If there is no previous tree node, the PrevNode property returns a null reference (Nothing in Visual Basic).

property Node.PrevVisibleNode as Node

Gets the previous visible tree node.

Type	Description
Node	A Node object that indicates the previous visible node.

Use the PrevVisibleNode property to get the previous visible node. Use the NextVisibleNode property to get the next visible node. Use the Visible property to hide a node. Use the FirstVisibleNode property to get the first visible node in the control's client area. If there is no previous tree node, the PrevVisibleNode property returns a null reference (Nothing in Visual Basic).

property Node.Selected as Boolean

Specifies whether the node is selected.

Type	Description
Boolean	A boolean expression that indicates whether the node is selected.

Use the Selected property to select a node. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to get the selected node by its index. The ClearSel method clears the collection of selected nodes. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. Use the FocusNode property to retrieve the focused node. Use the SingleSel property to specify whether the control support single or multiple selection. The control fires the SelectionChanged event when user changes the selection.

The following VB sample selects the node over the cursor as soon as the user moves the cursor over the control:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
   With XMLGrid1
      Dim n As EXMLGRIDLibCtl.Node
      Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
      If Not n Is Nothing Then
         n.Selected = True
      End If
   End With
End Sub
```

The following C++ sample selects the node over the cursor as soon as the user moves the cursor over the control:

```
#include "Node.h"
void OnMouseMoveXmlgrid1(short Button, short Shift, long X, long Y)
{
   CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
   if ( node.m_lpDispatch != NULL )
      node.SetSelected( TRUE );
}
```

The following VB.NET sample selects the node over the cursor as soon as the user moves the cursor over the control:

```
Private Sub AxXMLGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent) Handles
AxXMLGrid1.MouseMoveEvent
    With AxXMLGrid1
        Dim n As EXMLGRIDLib.Node = .get_NodeFromPoint(e.x, e.y)
        If Not n Is Nothing Then
            n.Selected = True
        End If
    End With
End Sub
```

The following C# sample selects the node over the cursor as soon as the user moves the cursor over the control:

```
private void axXMLGrid1_MouseMoveEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent e)
{
    EXMLGRIDLib.Node node = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
    if (node != null)
        node.Selected = true;
}
```

The following VFP sample selects the node over the cursor as soon as the user moves the cursor over the control:

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

with thisform.XMLGrid1
    n = .NodeFromPoint(x, y )
    if ( !isnull(n) )
        n.Selected = .t.
    endif
endwith
```

property Node.ToolTip as String

Specifies the node's tooltip.

Type	Description
String	A String expression that indicates the node's tooltip.

Use the ToolTip property to assign a tooltip to a node. The node's tooltip shows up when the cursor hovers the node. Use the ToolTipDelay property to specify the time in ms that passes before the ToolTip appears. 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. The ShowToolTip method shows programmatically the control's tooltip. Use the Background(exToolTipBackColor) property indicates the tooltip's background color.

The ToolTip property supports built-in HTML format like follows:

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

the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.

- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with $^{\text{subscript}}$
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

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

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

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

property Node.ToolTipTitle as String

Specifies the node's title for its tooltip.

Type	Description
String	A String expression that defines the title for the node's tooltip.

By default, the ToolTipTitle property is empty string. The ToolTipTitle defines the title for the node's tooltip. Use the ToolTip property to assign a tooltip to a node. The node's tooltip shows up when the cursor hovers the node. Use the ToolTipDelay property to specify the time in ms that passes before the ToolTip appears. 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. The ShowToolTip method shows programmatically the control's tooltip.

property Node.UserData as Variant

Associates an extra data to the node.

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

Use the UserData property to associate an extra data to a node. Use the RemoveNode event to release any extra data associated to a node.

property Node.Value as Variant

Specifies the value of the node.

Type	Description
Variant	A Value expression that indicates the node's value.

The node's Value property is displayed only if the node contains no child nodes. Use the Name property to assign a caption to the node. Use the Editor property to assign an editor to a node. The control fires the Change event when the user changes the node's value if the node has no child nodes, or the node's name if the node has child nodes. Use the Add method to specify the node's name and value at adding time. Use the UserData property to assign an extra data to a node. Use the BackColorValue property to specify the node's value background color. Use the ForeColorValue property to specify the node's value foreground color

If the node has no child nodes, and it has an editor assigned, the editor display the node's value based on the editor and the node's value. For instance, if you have a drop down list editor, the control displays the associated item to the node's value in the editor's list of items. If the node has no editor assigned the Value property indicates the text being displayed in the node's value area. The text supports built-in HTML format like described bellow.

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

the color in hexa values.

- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{subscript}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a

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

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

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

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

property Node.Visible as Boolean

Specifies whether a node is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether a node is visible or hidden.

Use the Visible property to hide a node. Use the Remove method to remove a node. Use the NodeHeight property to specify the height of the nodes. Use the Position property to specify the node's position inside the nodes collection. Use the FirstVisibleNode property to get the first visible node in the control's client area. Use the NextVisibleNode property to get the next visible node. Use the PrevVisibleNode property to get the previous visible node.

The following sample displays the list of visible nodes as they are displayed:

```
With XMLGrid1
   Dim n As EXMLGRIDLibCtl.Node, i As Long
   i = 0
   Set n = .NodeByPosition(i)
   While Not n Is Nothing
      Debug.Print n.Name
      i = i + 1
      Set n = .NodeByPosition(i)
   Wend
End With
```

Nodes object

The Nodes object holds a collection of Node objects. The Nodes object holds the control's nodes collection. Use the Nodes property to access the Nodes collection. Use the Editors property to access to the control's editors collection. The Nodes collection supports the following properties and methods:

Name	Description
Add	Adds a child node and returns a reference to the newly created object.
Clear	Removes all objects in a collection.
Count	Returns the number of objects in a collection.
Item	Returns a specific node of the Nodes collection.
ItemByID	Returns a node giving its unique identifier.
ItemByPosition	Retrieves a node giving its position.
Parent	Retrieves the node's parent.
Remove	Removes a specific member from the Nodes collection.
RemoveByID	Removes a member giving its unique identifier.

method Nodes.Add (Name as String, [Value as Variant], [Key as Variant])

Adds a child node and returns a reference to the newly created object.

Type	Description
Name as String	A string expression that indicates the name of the node being inserted.
Value as Variant	A Variant expression that indicates the value of the node being inserted.
Key as Variant	A string or long expression that indicates the key of the node being inserted.

Return	Description
Node	A Node object being created.

Use the Add method to add new nodes to the control. Use the LoadXML method to load XML documents. Use the Nodes property to access the node's child nodes collection. Use the Editors property to access the control's Editors collection. The control fires the AddNode event when a new node is inserted to the control's nodes collection. The Name and Value parameters support built-in HTML format. Use the Parent property to get the node's parent. The AllowDuplicateEntries property returns or sets a value that specifies whether the control supports nodes with the same key (duplicates).

The following sample adds few nodes to the control's nodes collection.

```
Private Sub Form_Load()
  With XMLGrid1
    .BeginUpdate

    With .Nodes
      With .Add("Root").Nodes
        .Add "Child 1", "text1"
        .Add "Child 2", "text2"
      End With
    End With
    .EndUpdate
  End With
End Sub
```

method Nodes.Clear ()

Removes all objects in a collection.

Type	Description

Use the Clear method to clear the nodes collection.

property Nodes.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	A long expression that retrieves the number of elements in the collection.

The Count property gets the number of nodes in the collection. Use the Item property to access a Node object.

The following sample shows how to enumerate the nodes in the collection:

```
Dim n As EXMLGRIDLibCtl.Node
For Each n In XMLGrid1.Nodes
   Debug.Print n.Key
Next
```

or

```
Dim i As Long
With XMLGrid1.Nodes
   For i = 0 To .Count - 1
      Debug.Print .Item(i).Key
   Next
End With
```

The following sample enumerates all visible nodes in the control:

```
With XMLGrid1
   Dim n As EXMLGRIDLibCtl.Node, i As Long
   i = 0
   Set n = .NodeByPosition(i)
   While Not n Is Nothing
      Debug.Print n.Name
      i = i + 1
      Set n = .NodeByPosition(i)
   Wend
End With
```

property Nodes.Item (Index as Variant) as Node

Returns a specific node of the Nodes collection.

Type	Description
Index as Variant	A string expression that indicates the key of the node being searched, or a long expression that indicates the index of node being accessed.
Node	A Node object being accessed.

Use the Item property to access a Node object giving its index or its key. The Count property gets the number of nodes in the collection. Use the ItemByPosition property to enumerate the root nodes as they are displayed. The Name property indicates the name of the node, where the Value property specifies the node's value. The FirstNode property specifies the first child node. Use the NextNode property to specify the next child node.

The following sample shows how to enumerate the nodes in the collection:

```
Dim n As EXMLGRIDLibCtl.Node
For Each n In XMLGrid1.Nodes
   Debug.Print n.Key
Next
```

or

```
Dim i As Long
With XMLGrid1.Nodes
   For i = 0 To .Count - 1
      Debug.Print .Item(i).Key
   Next
End With
```

The following sample enumerates all visible nodes in the control:

```
With XMLGrid1
   Dim n As EXMLGRIDLibCtl.Node, i As Long
   i = 0
   Set n = .NodeByPosition(i)
   While Not n Is Nothing
      Debug.Print n.Name
      i = i + 1
```

```
    Set n = .NodeByPosition(i)
  Wend
End With
```

property Nodes.ItemByID (ID as Variant) as Node

Returns a node giving its unique identifier.

Type	Description
ID as Variant	A String expression that specifies the unique identifier of the node, previously returned by the ID property of the Node.
Node	A Node object being returned.

The ItemByID property returns a node giving its unique identifier. The ID property is generated by the control, to identify uniquely a node within the Nodes collection. The RemoveByID method removes a node giving its unique identifier.

property Nodes.ItemByPosition (Position as Long) as Node

Retrieves a node giving its position.

Type	Description
Position as Long	A Long expression that indicates the position of the node being requested. The Position expression is 0 based, where 0 indicates the first visible node.
Node	A Node object that indicates the node at position

Use the ItemByPosition property to retrieve a node by its position. The Count property counts the number of nodes in the collection. The Name property indicates the name of the node, where the Value property specifies the node's value. The FirstNode property specifies the first child node. Use the NextNode property to specify the next child node. Use the ItemByPosition property to enumerate the root nodes as they are displayed. Use the Item property to retrieve a node giving its key or its index.

The following VB sample enumerates all nodes in the control as they are displayed (including child nodes too):

```
Private Sub enumerate(ByVal x As EXMLGRIDLibCtl.XMLGrid)
   With x.Nodes
      Dim i As Long
      For i = 0 To .Count - 1
         enumNodes .ItemByPosition(i)
      Next
   End With
End Sub

Private Sub enumNodes(ByVal n As EXMLGRIDLibCtl.Node)
   Dim c As EXMLGRIDLibCtl.Node
   Debug.Print n.Name
   Set c = n.FirstNode
   While Not c Is Nothing
      enumNodes c
      Set c = c.NextNode
   Wend
End Sub
```

property Nodes.Parent as Node

Retrieves the node's parent.

Type	Description
Node	A Node object that indicates the owner node of the Nodes collection.

Use the Parent property to get the owner node of the Nodes collection. If the Parent property points to nothing, the Nodes collection belongs to the control, and can be accessed using the Nodes property.

method Nodes.Remove (Index as Variant)

Removes a specific member from the Nodes collection.

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

Use the Remove method to remove a node from the control's nodes collection. The RemoveNode event is fired each time a node is removed. Use the Clear method to clear the control's nodes collection, or the child nodes collection. Use the Visible property to hide a node.

method Nodes.RemoveByID (Index as Variant)

Removes a member giving its unique identifier.

Type	Description
Index as Variant	A String expression that specifies the unique identifier of the node, previously returned by the ID property of the Node.

The RemoveByID method removes a node giving its unique identifier. The ID property is generated by the control, to identify uniquely a node within the Nodes collection. The ItemByID property gets the node giving its identifier.

OleEvent object

The OleEvent object holds information about an event fired by an ActiveX contro. The UserEditorOleEvent event uses the same type of the object to hold information about an OLE event.

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

property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

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

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

```
Private Sub XMLGrid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXMLGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Node As
EXMLGRIDLibCtl.INode)
  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
```

property OleEvent.ID as Long

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

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

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

property OleEvent.Name as String

Retrieves the original name of the fired event.

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

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 Name property indicates the name of the OLE event being fired.

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

```
Private Sub XMLGrid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXMLGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Node As
EXMLGRIDLibCtl.INode)
  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
```

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 a string expression that indicates the argument's name.
OleEventParam	An OleEventParam object that contains the name and the value for the argument.

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

```
Private Sub XMLGrid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXMLGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Node As
EXMLGRIDLibCtl.INode)
  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
```

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
Name	Retrieves the name of the event's parameter.
Value	Retrieves the value of the event's parameter.

property OleEventParam.Name as String

Retrieves the name of the event's parameter.

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

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

```
Private Sub XMLGrid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXMLGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Node As
EXMLGRIDLibCtl.INode)
  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
```

property OleEventParam.Value as Variant

Retrieves the value of the event's parameter.

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

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

```
Private Sub XMLGrid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXMLGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Node As
EXMLGRIDLibCtl.INode)
  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
```

XMLGrid 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: {AC7F976E-48C3-4B0B-B952-45D92DFE7F3E}. The object's program identifier is: "Exontrol.XMLGrid". The /COM object module is: "EXMLGrid.dll"

Exontrols new eXMLGrid control provides an innovative grid view look and handles data in XML fashion way. It provides swift and robust performance and a wide range of formatting features never seen on other grids. The eXMLGrid component can be seen as a generalized tree control that allows resizing the node's indentation at runtime. Use the Nodes property to access the control's nodes collection. Use the Editors property to access the control's editors collection. The eXMLGrid 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 component supports the following properties and methods:

Name	Description
AlignChildContent	Indicates whether the control aligns the child content.
AllowDuplicateEntries	Returns or sets a value that specifies whether the control supports nodes with the same key (duplicates).
AnchorFromPoint	Retrieves the identifier of the anchor from point.
Appearance	Retrieves or sets the control's appearance.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoEdit	Specifies whether the node may be edited when it has the focus.
AutoSearch	Enables or disables the incremental searching feature.
BackColor	Specifies the control's background color.
Background	Returns or sets a value that indicates the background color for parts in the control.
BeginUpdate	Maintains performance when nodes are added to the control one at a time. This method prevents the control from painting until the EndUpdate method is called.
BorderHeight	Sets or retrieves a value that indicates the border height of the control.
BorderWidth	Sets or retrieves a value that indicates the border width of the control.
ClearSel	Clears the collection of the selected nodes.
CollapseAll	Collapses all the nodes.

Edit	Edits the focused node.
Editing	Specifies the window's handle of the built-in editor while the control is running in edit mode.
Editors	Retrieves the control's Editors collection.
Enabled	Enables or disables the control.
EndUpdate	Resumes painting the control after painting is suspended by the BeginUpdate method.
EnsureVisibleNode	Ensures that the node is visible, expanding tree nodes and scrolling the tree view control as necessary.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
ExpandAll	Expands all the nodes.
ExpandBarVisible	Specifies whether the control's expand bar is visible or hidden.
ExpandButtons	Adds a button to the left side of each parent item. The user can click the button to expand or collapse the child nodes as an alternative to double-clicking the parent item.
ExpandButtonsCustom	Specifies the index of icons for +/- signs when the ExpandButtons property is exCustom.
ExpandOnDblClk	Specifies whether the node is expanded or collapsed if the user dbl clicks the node.
ExpandOnKeys	Specifies a value that indicates whether the control expands or collapses a node when user presses arrow keys.
ExpandOnSearch	Expands nodes automatically while user types characters to search for a specific node.
FilterBarPrompt	Specifies the caption to be displayed when the filter pattern is missing.
FilterBarPromptPattern	Specifies the pattern for the filter prompt.
FilterBarPromptType	Specifies the type of the filter prompt.
FilterBarPromptVisible	Shows or hides the filter prompt.
FirstVisibleNode	Gets the first visible tree node in the tree view control.
FocusNode	Specifies the focus node.
Font	Retrieves or sets the control's font.

ForeColor	Specifies the control's foreground color.
FormatAnchor	Specifies the visual effect for anchor elements in HTML captions.
GridLines	Specifies whether the control renders grid lines.
GridLinesColor	Specifies a value that indicates the grid line color.
HideSelection	Specifies whether the selection is hidden when control loses the focus.
HitTest	Determines which portion of a node is at specified point.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
Images	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
ImageSize	Retrieves or sets the size of icons the control displays..
Layout	Saves or loads the control's layout, such selected nodes, scroll position, and so on.
LevelWidth	Returns or sets a value that indicates the width of the level.
LoadXML	Loads an XML document from the specified location, using MSXML parser.
MoveCursorOnCollapse	Moves the cursor when a node is collapsed using the mouse.
NodeByPosition	Retrieves a node giving its position.
NodeFromPoint	Retrieves the node's from point.
NodeHeight	Sets or gets a value that indicates the node's height.
Nodes	Retrieves the Nodes collection.
OLEDrag	Causes a component to initiate an OLE drag/drop operation.
OLEDropMode	Returns or sets how a target component handles drop operations
Picture	Retrieves or sets a graphic to be displayed in the control.
PictureDisplay	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background
ReadOnly	Specifies whether the control is read only.
Refresh	Refreshes the control.

ReplaceIcon	Adds a new icon, replaces an icon or clears the control's image list.
ResizeToFit	Resizes the control's level (and the next ones) so its content its fully visible.
SaveXML	Saves the control's content as XML document to the specified location, using the MSXML parser.
Scroll	Scrolls the control's content.
ScrollBars	Specifies the type of scroll bars that control has.
ScrollButtonHeight	Specifies the height of the button in the vertical scrollbar.
ScrollButtonWidth	Specifies the width of the button in the horizontal scrollbar.
ScrollFont	Retrieves or sets the scrollbar's font.
ScrollHeight	Specifies the height of the horizontal scrollbar.
ScrollOrderParts	Specifies the order of the buttons in the scroll bar.
ScrollPartCaption	Specifies the caption being displayed on the specified scroll part.
ScrollPartCaptionAlignment	Specifies the alignment of the caption in the part of the scroll bar.
ScrollPartEnable	Indicates whether the specified scroll part is enabled or disabled.
ScrollPartVisible	Indicates whether the specified scroll part is visible or hidden.
ScrollPos	Specifies the vertical/horizontal scroll position.
ScrollThumbSize	Specifies the size of the thumb in the scrollbar.
ScrollToolTip	Specifies the tooltip being shown when the user moves the scroll box.
ScrollWidth	Specifies the width of the vertical scrollbar.
Search	Searches for a node.
SelBackColor	Specifies the selection's background color.
SelBackColorChild	Specifies the selection's background color on the value section.
SelBackColorCollapse	Specifies the selection's background color, when the node is collapsed.
SelBackMode	Retrieves or sets a value that indicates whether the selection is transparent or opaque.

SelectAll	Selects all nodes. The property is available only if the SingleSel property is False.
SelectCount	Specifies the number of selected node.
SelectedNode	Retrieves the selected node.
SelForeColor	Specifies the selection foreground's color.
SelForeColorChild	Specifies the selection's background color on the value section.
ShowFocusRect	Retrieves or sets a value indicating whether the control draws a thin rectangle around the focused item.
ShowImageList	Specifies whether the control's image list window is visible or hidden.
ShowPartialParent	Specifies where a partial-visible parent shows its content.
ShowToolTip	Shows the specified tooltip at given position.
SingleSel	Specifies whether the control supports single or multiple selection.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipPopDelay	Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.
ToolTipWidth	Specifies a value that indicates the width of the tooltip window, in pixels.
UnselectAll	Unselects all nodes. The property is available only if the SingleSel property is False.
UseVisualTheme	Specifies whether the control uses the current visual theme to display certain UI parts.
Version	Retrieves the control's version.
VisibleLevelCount	Returns a value that indicates the number of visible levels in the tree control.
VisibleNodeCount	Specifies the number of visible nodes.
VisualAppearance	Retrieves the control's appearance.
VisualDesign	Invokes the control's VisualAppearance designer.

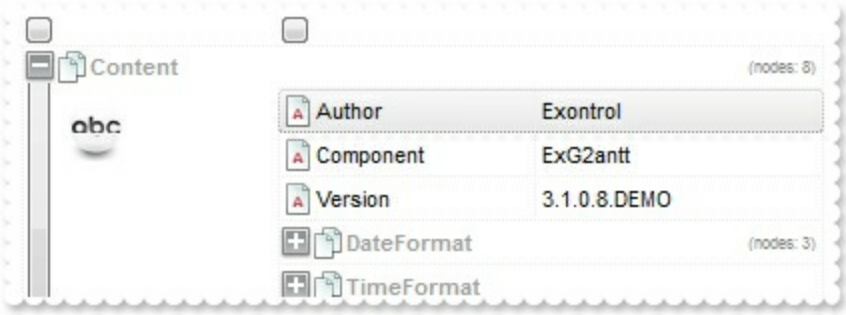
property XMLGrid.AlignChildContent as Boolean

Indicates whether the control aligns the child content.

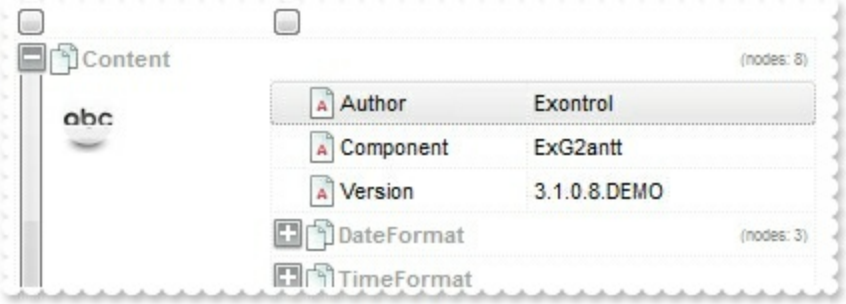
Type	Description
Boolean	A Boolean expression that indicates whether the control aligns the child content.

By default, the AlignChildContent property is False, which indicates the child content is not aligned (icons, text, expanding buttons, get aligned).

The following screen shot shows the control's content (AlignChildContent property is False, by default)



The following screen shot shows the control's content (AlignChildContent property is True)



property XMLGrid.AllowDuplicateEntries as Boolean

Returns or sets a value that specifies whether the control supports nodes with the same key (duplicates).

Type	Description
Boolean	A Boolean expression that specifies whether the control allows adding nodes with the same key.

By default, the AllowDuplicateEntries property is False, which indicates that nodes with the same key can not be added. Use the AllowDuplicateEntries property on True, to allow adding new nodes with the same key. You can change the AllowDuplicateEntries property only, if the control's Nodes collection is empty. The Add method of Nodes collection adds a new node to the Nodes collection.

property XMLGrid.AnchorFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as String

Retrieves the identifier of the anchor from point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in client coordinates.
String	A String expression that specifies the identifier (id) of the anchor element from the point, or empty string if there is no anchor element at the cursor.

Use the AnchorFromPoint property to determine the identifier of the anchor from the point. Use the <a id;options> anchor elements to add hyperlinks to cell's caption. The control fires the AnchorClick event when the user clicks an anchor element. Use the ShowToolTip method to show the specified tooltip at given or cursor coordinates. The MouseMove event is generated continually as the mouse pointer moves across the control.

The following VB sample displays (as tooltip) the identifier of the anchor element from the cursor:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With XMLGrid1
    .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 AxXMLGrid1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As AxEXXMLGRIDLib._IXMLGridEvents_MouseMoveEvent) Handles AxXMLGrid1.MouseMoveEvent
  With AxXMLGrid1
    .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 axXMLGrid1_MouseMoveEvent(object sender,
AxEXXMLGRIDLib._IXMLGridEvents_MouseMoveEvent e)
{
   axXMLGrid1.ShowToolTip(axXMLGrid1.get_AnchorFromPoint(-1, -1));
}
```

The following C++ sample displays (as tooltip) the identifier of the anchor element from the cursor:

```
void OnMouseMoveXMLGrid1(short Button, short Shift, long X, long Y)
{
   COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
   m_xmlGrid.ShowToolTip( m_xmlGrid.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 .XMLGrid1
      .ShowToolTip(.AnchorFromPoint(-1, -1))
   EndWith
endwith
```

property XMLGrid.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's border style.

Use the Appearance property to define the control's border style. Use the Appearance property to hide the control borders.

method XMLGrid.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 XMLGrid1_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 XMLGrid.AutoEdit as Boolean

Specifies whether the node may be edited when it has the focus.

Type	Description
Boolean	A boolean expression that indicates whether the control starts editing the focused node.

By default, the AutoEdit property is True. The AutoEdit property has no effect if the focused node has no editor assigned. Use the Editor property to assign an editor to a node. Use the Add method to add new type of editors to the control. Use the Edit method to programmatically edit the focused node, when AutoEdit property is False. Use the Editing property to specify whether the control is running in edit mode.

The following sample starts editing a node as soon as user presses the F2 key:

```
Private Sub XMLGrid1_KeyDown(KeyCode As Integer, Shift As Integer)
  With XMLGrid1
    If .Editing = 0 Then
      If KeyCode = vbKeyF2 Then
        .Edit
      End If
    End If
  End With
End Sub
```

property XMLGrid.AutoSearch as AutoSearchEnum

Enables or disables the incremental searching feature.

Type	Description
AutoSearchEnum	An AutoSearchEnum expression that indicates the kind of searching that control performs when user types characters.

By default, the AutoSearch property is exStartWith. Use the AutoSearch property to define a 'contains' incremental search. If the AutoSearch property is exContains, the control searches for nodes that contain the typed characters. Use the ExpandOnSearch property to expand nodes automatically while user types characters to search for a specific node. The Search property searches programmatically for for a node.

property XMLGrid.BackColor as Color

Specifies the control's background color.

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

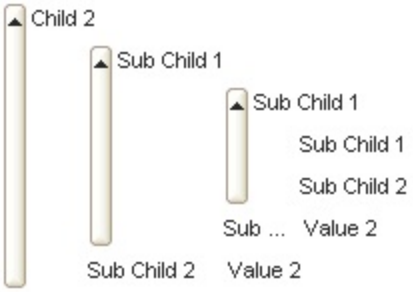
Use the BackColor property to specify the control's background color. Use the ForeColor property to specify the control's foreground color. Use the SelBackColor, SelForeColor, SelBackColorChild and SelForeColorChild properties to customize the colors for selected nodes. Use the BackColor property to specify the node's background color. Use the BackColorChild property to specify the background color for child nodes.

property XMLGrid.Background(Part as BackgroundPartEnum) as Color

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

Type	Description
Part as BackgroundPartEnum	A BackgroundPartEnum expression that indicates 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 Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

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



The following VB sample changes the visual appearance for the expand buttons. The sample uses the "▢" for up state, and "▢"

```
With XMLGrid1
  With .VisualAppearance
    .Add &H12, App.Path + "\expandu.ebn"
    .Add &H13, App.Path + "\expandd.ebn"
  End With
  .Background(exExpandButtonUp) = &H12000000
  .Background(exExpandButtonDown) = &H13000000
End With
```

The following C++ sample changes the visual appearance for the expand buttons:

```
#include "Appearance.h"
m_xmlgrid.GetVisualAppearance().Add( 0x12,
COleVariant(_T("D:\\Temp\\EXMLGrid.Help\\expandu.ebn")) );
m_xmlgrid.GetVisualAppearance().Add( 0x13,
COleVariant(_T("D:\\Temp\\EXMLGrid.Help\\expandd.ebn")) );
m_xmlgrid.SetBackground( 0, 0x12000000 );
m_xmlgrid.SetBackground( 1, 0x13000000 );
```

The following VB.NET sample changes the visual appearance for the expand buttons:

```
With AxXMLGrid1
   With .VisualAppearance
      .Add(&H12, "d:\temp\EXMLGrid.Help\expandu.ebn")
      .Add(&H13, "d:\temp\EXMLGrid.Help\expandd.ebn")
   End With
   .set_Background(EXMLGRIDLib.BackgroundPartEnum.exExpandButtonUp, &H12000000)
   .set_Background(EXMLGRIDLib.BackgroundPartEnum.exExpandButtonDown,
&H13000000)
End With
```

The following C# sample changes the visual appearance for the expand buttons:

```
axXMLGrid1.VisualAppearance.Add(0x12, "d:\\temp\\EXMLGrid.Help\\expandu.ebn");
axXMLGrid1.VisualAppearance.Add(0x13, "d:\\temp\\EXMLGrid.Help\\expandd.ebn");
axXMLGrid1.set_Background(EXMLGRIDLib.BackgroundPartEnum.exExpandButtonUp,
0x12000000);
axXMLGrid1.set_Background(EXMLGRIDLib.BackgroundPartEnum.exExpandButtonDown,
0x13000000);
```

The following VFP sample changes the visual appearance for the expand buttons:

```
With thisform.XMLGrid1
   With .VisualAppearance
      .Add(18, "D:\Temp\EXMLGrid.Help\expandu.ebn")
      .Add(19, "D:\Temp\EXMLGrid.Help\expandd.ebn")
   EndWith
   .Background(0) = 301989888
```

```
    .Background(1) = 318767104
EndWith
```

where the 301989888 value is the hexa representation for 0x12000000, and 318767104 is 0x13000000.

method XMLGrid.BeginUpdate ()

Maintains performance when items are added to the control one at a time. This method prevents the control from painting until the EndUpdate method is called.

Type	Description

The BeginUpdate method prevents the control from painting until the EndUpdate method is called. Use BeginUpdate and EndUpdate statement each time when the control requires more changes. Using the BeginUpdate and EndUpdate methods increase the speed of changing the control properties by preventing it from painting during changing. Use the Refresh method to refresh the control.

The sample adds several nodes to the control and prevents painting the control, while adding new nodes :

```
With XMLGrid1
   .BeginUpdate
      With .Nodes
         Dim i As Long
         For i = 1 To 100
            .Add "Child <b>" & i & "</b>"
         Next
      End With
   .EndUpdate
End With
```

property XMLGrid.BorderHeight as Long

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

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

Use the BorderWidth, BorderHeight property to specify the control's border size. By default, the BorderHeight property is 2 pixels.

property XMLGrid.BorderWidth as Long

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

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

Use the BorderWidth, BorderHeight property to specify the control's border size. By default, the BorderWidth property is 2 pixels.

method XMLGrid.ClearSel ()

Clears the collection of the selected nodes.

Type	Description

The ClearSel method clears the collection of selected nodes. Use the Selected property to select nodes. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to get the selected node by its index. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The control fires the SelectionChanged event when user changes the selection.

method XMLGrid.CollapseAll ()

Collapses all the nodes.

Type	Description

Use the CollapseAll method to collapse all nodes in the control. Use the ExpandAll method to expand all nodes in the control. Use the Expanded property to expand or collapse a node. Use the ExpandAll method to expand all child nodes. Use the CollapseAll method to collapse all child nodes.

method XMLGrid.Edit ([Options as Variant])

Edits the focused node.

Type	Description
Options as Variant	Reserved.

Use the Edit method to programmatically edit the focused node. Use the FocusNode property to specify the control's focused node. Use the Selected property to changes the selection. When user changes the selection the focused node is moved too. Use the ShowFocusRect property to mark focused node with a thin rectangle. Use the AutoEdit property to specify whether the control starts editing a cell as soon as the user moves the focused node. Use the Editor property to assign an editor to a node. Use the Editing property to check whether the control is running in the edit mode.

The edit events are fired in the following order:

1. Edit event. Prevents editing nodes, before showing the node's editor.
2. EditOpen event. The edit operation started, the node's editor is shown. The Editing property gives the window's handle of the built-in editor being shown.
3. Change event. The Change event is fired only if the user types ENTER key, the user selects a new value from a predefined data list, or focus a new node.
4. EditClose event. The node's editor is hidden and closed.

The following sample starts editing a node as soon as user presses the F2 key:

```
Private Sub XMLGrid1_KeyDown(KeyCode As Integer, Shift As Integer)
  With XMLGrid1
    If .Editing = 0 Then
      If KeyCode = vbKeyF2 Then
        .Edit
      End If
    End If
  End With
End Sub
```

property XMLGrid.Editing as Long

Specifies the window's handle of the built-in editor while the control is running in edit mode.

Type	Description
Long	A long expression that indicates the window's handle for the built-in editor that's focused while the control is running in the edit mode.

Use the Editing property to check whether the control is in edit mode. Use the Editing property to get the window's handle for the built-in editor while editing. Use the Edit method to start editing the focused cell. Use the EditType property to define the type of the editor. Use the ReadOnly property to make the control read only. Use the Editor property to assign an editor to a node.

The edit events are fired in the following order:

1. Edit event. Prevents editing nodes, before showing the node's editor.
2. EditOpen event. The edit operation started, the node's editor is shown. The Editing property gives the window's handle of the built-in editor being shown.
3. Change event. The Change event is fired only if the user types ENTER key, the user selects a new value from a predefined data list, or focus a new node.
4. EditClose event. The node's editor is hidden and closed.

property XMLGrid.Editors as Editors

Retrieves the control's Editors collection.

Type	Description
Editors	An Editors object that holds the collection of Editor objects.

Use the Editors property to access the control's editors collection. The control supports several type of editors like described in the EditTypeEnum enumeration. Use the Add method to add new type of editors to the control. Use the Editor property to assign an editor to a node. Use the EditType property to specify the type of editor being used. Use the Editing property to check whether the control is running in edit mode.

The following sample adds a spin editor to a node:

```
With XMLGrid1
  .BeginUpdate
    With .Editors
      With .Add("Spin")
        .ButtonWidth = 18
        .EditType = SpinType
        .AddButton "A", 1
        .AddButton "B", 1, RightAlignment
      End With
    End With
    With .Nodes
      With .Add("Spin", 1)
        .Editor = "Spin"
      End With
    End With
  .EndUpdate
End With
```

property XMLGrid.Enabled as Boolean

Enables or disables the control.

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

Use the Enabled property to disable the control. Use the ReadOnly property to prevent users changing the control's content. Use the Locked property to lock or unlock an editor.

method XMLGrid.EndUpdate ()

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

Type	Description

The BeginUpdate method prevents the control from painting until the EndUpdate method is called. Use BeginUpdate and EndUpdate statement each time when the control requires more changes. Using the BeginUpdate and EndUpdate methods increase the speed of changing the control properties by preventing it from painting during changing. Use the Refresh method to refresh the control.

The sample adds several nodes to the control and prevents painting the control, while adding new nodes :

```
With XMLGrid1
  .BeginUpdate
    With .Nodes
      Dim i As Long
      For i = 1 To 100
        .Add "Child <b>" & i & "</b>"
      Next
    End With
  .EndUpdate
End With
```

method XMLGrid.EnsureVisibleNode (Node as Variant)

Ensures that the node is visible, expanding tree nodes and scrolling the tree view control as necessary.

Type	Description
Node as Variant	A Node object being made visible.

Call the EnsureVisibleNode method to ensure that a control's node is visible. Use the NodeFromPoint property to get the node from point. If necessary, the method expands the parent node or scrolls the xml grid view control so that the node is visible.

property XMLGrid.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 XMLGrid.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed

Return	Description
Variant	A Variant expression that indicates the result after executing the Template.

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

For instance, the following sample retrieves the count of nodes:

```
Debug.Print XMLGrid.ExecuteTemplate("Nodes.Count")
```

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

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

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

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

An 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 XMLGrid.ExpandAll ()

Expands all the nodes.

Type	Description

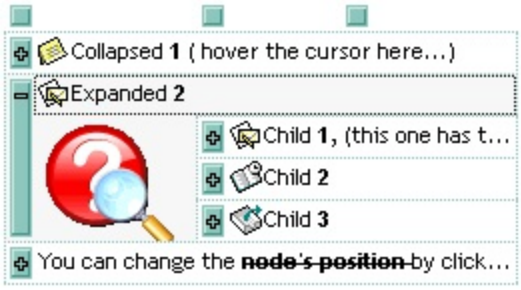
Use the ExpandAll method to expand all nodes in the control. Use the CollapseAll method to collapse all nodes in the control. Use the Expanded property to expand or collapse a node. Use the ExpandAll method to expand all child nodes. Use the CollapseAll method to collapse all child nodes.

property XMLGrid.ExpandBarVisible as Boolean

Specifies whether the control's expand bar is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the control's expand bar is visible or hidden.

By default, the ExpandBarVisible property is False. Use the ExpandBarVisible property to show the control's expand bar. The expand bar displays a button for each level found. Clicking a button on the expand bar makes the control to expand or collapse the nodes on the same level.



property XMLGrid.ExpandButtons as ExpandButtonEnum

Adds a button to the left side of each parent item. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item.

Type	Description
ExpandButtonEnum	An ExpandButtonEnum expression that indicates the type of expanding/collapsing buttons being displayed.

Use the ExpandButtons property to change the appearance for +/- buttons. Use the +/- buttons to expand or collapse nodes. Use the ExpandButtonsCustom property to assign icons for +/- buttons.

property XMLGrid.ExpandButtonsCustom(Expanded as Boolean) as Long

Specifies the index of icons for +/- signs when the ExpandButtons property is exCustom.

Type	Description
Expanded as Boolean	A boolean expression that indicates the expanding or collapsing button being changed.
Long	A long expression that indicates the index of icon being displayed.

Use the ExpandButtonsCustom property to assign icons for +/- buttons. Use the +/- buttons to expand or collapse nodes. Use the ExpandButtons property to change the appearance for +/- buttons. The ExpandButtonsCustom property has effect only if the ExpandButtons property is exCustom. Use the Images method to assign a list of icons to the control. Use the MoveCursorOnCollapse property to move the cursor when user collapses a node.

The following sample assigns different icons for +/- buttons:

```
With XMLGrid1
   .Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExm



   .ExpandButtons = exCustom
   .ExpandButtonsCustom(True) = 2
   .ExpandButtonsCustom(False) = 1
End With
```

property XMLGrid.ExpandOnDblClk as Boolean

Specifies whether the node is expanded or collapsed if the user dbl clicks the node.

Type	Description
Boolean	A boolean expression that indicates whether the node is expanded or collapsed when a node is double clicked.

Use the ExpandOnDblClk property to specify whether the control expands or collapses a node when user dbl clicks a node. Use the ExpandOnKeys property to allow users expand or collapse the nodes using the keyboard. The ExpandOnSearch property specifies whether the control expands nodes when incremental searching is on (AutoSearch property is different than 0) and user types characters when the control has the focus.

Use the Expanded property to expand or collapse a node. Use the ExpandAll method to expand all nodes in the control. Use the CollapseAll method to collapse all nodes in the control. Use the ExpandAll method to expand all child nodes. Use the CollapseAll method to collapse all child nodes.

property XMLGrid.ExpandOnKeys as Boolean

Specifies a value that indicates whether the control expands or collapses a node when user presses arrow keys.

Type	Description
Boolean	A boolean expression that indicates whether the control expands or collapses a node when user presses arrow keys.

Use the ExpandOnKeys property to specify whether the control expands or collapses a node when user presses arrow keys. By default, the ExpandOnKeys property is True. Use the ExpandOnDblClk property to specify whether the control expands or collapses a node when user dbl clicks a node. The ExpandOnSearch property specifies whether the control expands nodes when incremental searching is on (AutoSearch property is different than 0) and user types characters when the control has the focus.

property XMLGrid.ExpandOnSearch as Boolean

Expands nodes automatically while user types characters to search for a specific node.

Type	Description
Boolean	A boolean expression that indicates whether the control expands nodes automatically while user types characters to search for a specific node.

Use the ExpandOnSearch property to expand nodes while user types characters to search for nodes using incremental search feature. By default, the ExpandOnSearch property is True. Use the AutoSearch property to enable or disable incremental searching feature. The ExpandOnSearch property has no effect when the AutoSearch property is False.

property XMLGrid.FilterBarPrompt as String

Specifies the caption to be displayed when the filter pattern is missing.

Type	Description
String	A string expression that indicates the HTML caption being displayed in the filter bar, when filter prompt pattern is missing. The FilterBarPromptPattern property specifies the pattern to filter the list using the filter prompt feature.

By default, the FilterBarPrompt property is "<fgcolor=808080>Start Filter...</fgcolor>". The FilterBarPromptPattern property specifies the pattern to filter the list using the filter prompt feature. Changing the FilterBarPrompt property won't change the current filter.

The FilterBarPrompt property supports HTML format as described here:

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

</dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.

- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with $^{\text{subscript}}$
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

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

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

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

The FilterBarPrompt property has effect only if:

- FilterBarPromptVisible property is True
- FilterBarPromptPattern property is Empty.

property XMLGrid.FilterBarPromptPattern as String

Specifies the pattern for the filter prompt.

Type	Description
String	A string expression that specifies the pattern to filter the list.

By default, the FilterBarPromptPattern property is empty. If the FilterBarPromptPattern property is empty, the filter bar displays the FilterBarPrompt property, if the FilterBarPromptVisible property is True. The FilterBarPromptPattern property indicates the patter to filter the list. The pattern may include wild characters if the FilterBarPromptType property is exFilterPromptPattern.

property XMLGrid.FilterBarPromptType as FilterPromptEnum

Specifies the type of the filter prompt.

Type	Description
FilterPromptEnum	A FilterPromptEnum expression that specifies how the items are being filtered.

By default, the FilterBarPromptType property is exFilterPromptContainsAll. The filter prompt feature allows you to filter the items as you type while the filter bar is visible on the bottom part of the list area. The Filter prompt feature allows at runtime filtering data on hidden columns too. Use the FilterBarPromptVisible property to show the filter prompt. Use the FilterBarPrompt property to specify the HTML caption being displayed in the filter bar when the filter pattern is missing. The FilterBarPromptPattern property specifies the pattern to filter the list.

The FilterBarPromptType property supports the following values:

- **exFilterPromptContainsAll**, The list includes the items that contains all specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptContainsAny**, The list includes the items that contains any of specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptStartWith**, The list includes the items that starts with any specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptEndWith**, The list includes the items that ends with any specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptPattern**, The filter indicates a pattern that may include wild characters to be used to filter the items in the list. The FilterBarPromptPattern property may include wild characters as follows:
  - '?' for any single character
  - '*' for zero or more occurrences of any character
  - '#' for any digit character
  - ' ' space delimits the patterns inside the filter

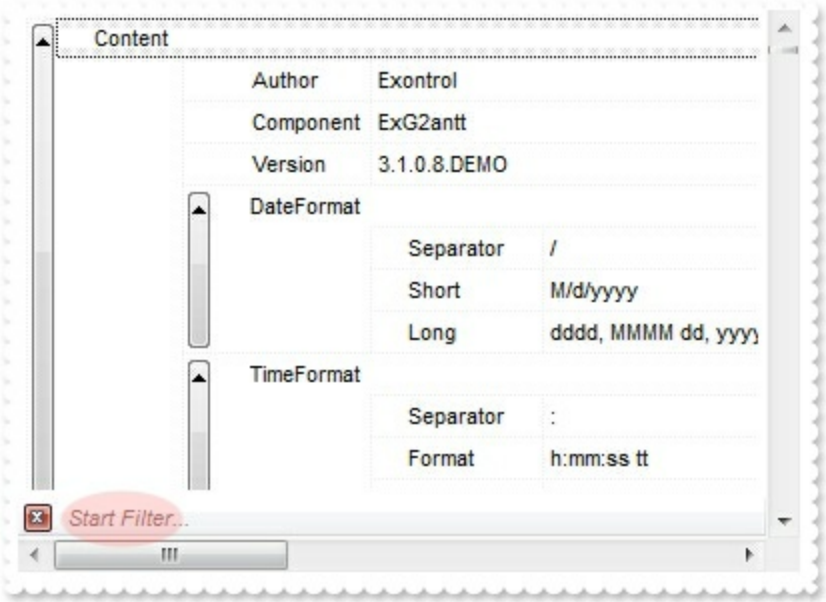
property XMLGrid.FilterBarPromptVisible as FilterBarVisibleEnum

Shows or hides the filter prompt.

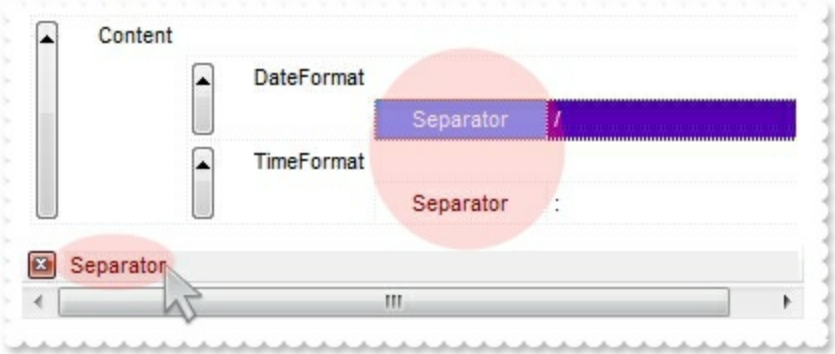
Type	Description
FilterBarVisibleEnum	A FilterBarVisibleEnum expression that specifies whether the control's filter-prompt is visible or hidden.

BY default, the FilterBarPromptVisible property is exFilterBarHidden. Use the FilterBarPromptVisible property to show and use the control's filter-prompt. The filter prompt feature allows you to filter the nodes as you type while the filter bar is visible on the bottom part of the list area. Use the FilterBarPrompt property to specify the HTML caption being displayed in the filter bar when the filter pattern is missing. The FilterBarPromptPattern property specifies the pattern to filter the list. The FilterBarPromptType property specifies how the filter is applied on node names and/or values.

The following screen show shows the filter prompt (FilterBarPromptVisible property is exFilterBarVisible):



The following screen show shows the list once the user types "Separator":



property XMLGrid.FirstVisibleNode as Node

Gets the first visible tree node in the tree view control.

Type	Description
Node	A Node object that's first visible node in the control's client area.

Use the FirstVisibleNode property to get the first visible node in the control's client area. Use the NodeByPosition property to access the node as they are displayed. Use the NextVisibleNode property to retrieve the next visible node. Use the NodeFromPoint property to get the node from cursor. Use the NextNode property to get the next sibling node. Use the Visible property to hide a node. Use the Position property to change the node's position inside the node's list of child nodes.

property XMLGrid.FocusNode as Node

Specifies the focus node.

Type	Description
Node	A Node object that indicates the focused node.

Use the FocusNode property to retrieve the focused node. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. Use the SelectionChanged event to notify your application when the focus is moved. Use the ShowFocusRect property to mark with a thin rectangle the focused node.

property XMLGrid.Font as IFontDisp

Retrieves or sets the control's font.

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

Use the Font property of an object to identify a specific Font object whose properties you want to use. Use the BackColor property to change the control's background color. Use the ForeColor property to specify the control's foreground color.

property XMLGrid.ForeColor as Color

Specifies the control's foreground color.

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

Use the ForeColor property to specify the control's foreground color. Use the BackColor property to specify the control's background color. Use the SelBackColor, SelForeColor, SelBackColorChild and SelForeColorChild properties to customize the colors for selected nodes. Use the BackColor property to specify the node's background color. Use the BackColorChild property to specify the background color for child nodes.

property XMLGrid.FormatAnchor(New as Boolean) as String

Specifies the visual effect for anchor elements in HTML captions.

Type	Description
New as Boolean	A Boolean expression that indicates whether to specify the anchors never clicked or anchors being clicked.
String	A String expression that indicates the HTMLformat to apply to anchor elements.

By default, the FormatAnchor(**True**) property is "<u><fgcolor=0000FF>#" that indicates that the anchor elements (that were never clicked) are underlined and shown in light blue. Also, the FormatAnchor(**False**) property is "<u><fgcolor=000080>#" that indicates that the anchor elements are underlined and shown in dark blue. The visual effect is applied to the anchor elements, if the FormatAnchor property is not empty. For instance, if you want to do not show with a new effect the clicked anchor elements, you can use the FormatAnchor(**False**) = "", that means that the clicked or not-clicked anchors are shown with the same effect that's specified by FormatAnchor(**True**). An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The **<a>** element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the AnchorClick event to notify that the user clicks an anchor element. This event is fired only if prior clicking the control it shows the hand cursor. The AnchorClick event carries the identifier of the anchor, as well as application options that you can specify in the anchor element. The hand cursor is shown when the user hovers the mouse on the anchor elements.

property XMLGrid.GridLines as GridLinesEnum

Specifies whether the control renders grid lines.

Type	Description
GridLinesEnum	A GridLinesEnum expression that indicates whether the control draws the grid lines.

The GridLines property indicates whether the control draws the grid lines. By default, the GridLines property is exDotLines. Use the GridLinesColor property to specify the color of the control's grid lines.

property XMLGrid.GridLinesColor as Color

Specifies a value that indicates the grid line color.

Type	Description
Color	A color expression that indicates the color for control's grid lines.

Use the GridLinesColor property to specify the color of the control's grid lines. The GridLines property indicates whether the control draws the grid lines. By default, the GridLinesColor property is &H8000000F&.

property XMLGrid.HideSelection as HideSelectionEnum

Specifies whether the selection is hidden when control loses the focus.

Type	Description
HideSelectionEnum	A HideSelectionEnum expression that indicates whether the selection is hidden when control loses the focus.

Use the HideSelection property to specify whether the control marks the selected nodes even if the control loses the focus. Use the SingleSel property to allow multiple selection. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node.

property XMLGrid.HitTest (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, Node as Node) as HitTestEnum

Determines which portion of a node is at specified point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates
Node as Node	A Node object where the cursor is.
HitTestEnum	A HitTestEnum expression that indicates the location of the cursor relative to the control's client area.

Call the HitTest method to determine the location of the specified point relative to the client area of a xml grid view control. Use the NodeFromPoint property to get the node from the cursor. Use the Name property to specify the name of the node.

The following VB sample displays the hit test code while user moves the mouse:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With XMLGrid1
    Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
    h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
    If Not h = 0 Then
      If (Not n Is Nothing) Then
        Debug.Print "Node = " & n.Name & " H = " & Hex(h)
      Else
        Debug.Print "H = " & Hex(h)
      End If
    End If
  End With
End Sub
```

The following C++ sample displays the hit test code while user moves the mouse:

```
#include "Node.h"
```

```
void OnMouseMoveXmlgrid1(short Button, short Shift, long X, long Y)
{
   CNode node; node.m_bAutoRelease = FALSE;
   long nHitTest = m_xmlgrid.GetHitTest( X, Y, &node.m_lpDispatch );
   if ( node.m_lpDispatch != NULL )
   {
      CString strFormat;
      strFormat.Format( "HitTest = 0x%04X, '%s' ", nHitTest, node.GetName() );
      OutputDebugString( strFormat );
   }
}
```

The following VB.NET sample displays the hit test code while user moves the mouse:

```
Private Sub AxXMLGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent) Handles
AxXMLGrid1.MouseMoveEvent
   With AxXMLGrid1
      Dim node As EXMLGRIDLib.Node = Nothing
      Dim hitTest As EXMLGRIDLib.HitTestEnum = .get_HitTest(e.x, e.y, node)
      If Not node Is Nothing Then
         Dim strMessage As String = "HitTest = " & hitTest.ToString() & " '" & node.Name &
"'"
         Debug.Write(strMessage)
      End If
   End With
End Sub
```

The following C# sample displays the hit test code while user moves the mouse:

```
private void axXMLGrid1_MouseMoveEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent e)
{
   EXMLGRIDLib.Node node = null;
   EXMLGRIDLib.HitTestEnum hitTest = axXMLGrid1.get_HitTest(e.x, e.y, out node );
   if (node != null)
   {
      String strMessage = "HitTest " + hitTest.ToString() + " '" + node.Name + "'";
```

```
        System.Diagnostics.Debug.Write(strMessage);
    }
}
```

The following VFP sample displays the hit test code while user moves the mouse:

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

with thisform.XMLGrid1
    local n, hitTest
    n = .SelectedNode(0)
    hitTest = .HitTest(x, y, @n )
    if ( !isnull(n) )
        wait window nowait "H:" + Str(hitTest) + " " + n.Name
    endif
endwith
```

property XMLGrid.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

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

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

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

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

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


A
B
△
A
+ B

property XMLGrid.hWnd as Long

Retrieves the control's window handle.

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

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

method XMLGrid.Images (Handle as Variant)

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

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

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

The control provides an image list window, that's displayed at design time. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ShowImageList property to hide the image list window, at design time. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list window. At runtime, the user can use the Images and ReplaceIcon method to change the Images collection. Use the Image property to assign an icon to a node. In case you are using the LoadXML method, the Image property of the Node indicates the type of XML node being added. Use the Picture property to assign a picture to a node. In case you are using the LoadXML method, use the Images method to add images to the control, so each type of element in your XML file, has a specific representation. The first icon in the Images collection indicates the NODE_ELEMENT type, the second icon in the Images collection indicates the NODE_ATTRIBUTE type, and so on.

The following sample shows how to replace the entire list of icons, using a Microsoft Image List control (ImageList1):

XMLGrid1.Images ImageList1.hImageList


```
With XMLGrid1
   .BeginUpdate
      .Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExm

      With .Nodes
```

```
        With .Add("Root")
            .Image = 1
        End With
    End With
  .EndUpdate
End With
```

property XMLGrid.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 XMLGrid.Layout as String

Saves or loads the control's layout, such selected nodes, scroll position, and so on.

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 width of the node's llevel). 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:

- widths of the visible levels
- vertical/horizontal scroll position
- filter options (if any)
- expanded/collapsed nodes
- selected nodes
- focused node

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 XMLGrid.LevelWidth(Level as Long) as Long

Returns or sets a value that indicates the width of the level.

Type	Description
Level as Long	A long expression that inidcates the level being resized. The 0 Level indicates the first level. The 1 Level indicates the second level and so on.
Long	A long expression that indicates the width of the level, in pixels.

Use the LevelWidth property to specify the level's width. The control fires the ResizeLevel event when user resizes a level. Use the Level property to get the node's level. Use the VisibleLevelCount property to specify the number of levels being displayed. You can use the ResizeToFit method to resize the visible levels to fit the visible node content.

The following sample specify a minimum width for the first level:

```
Private Sub XMLGrid1_ResizeLevel(ByVal Level As Long)
  If Level = 0 Then
    With XMLGrid1
      If .LevelWidth(Level) < 64 Then
        .LevelWidth(Level) = 64
      End If
    End With
  End If
End Sub
```

method XMLGrid.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 • string that indicates the file name, a URL, or a XML supplied string, • IStream, • SAFEARRAY, • 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. The control is emptied when the LoadXML method is called. During loading, the control fires the AddNode event for each XML node added to the control. For instance, this way, you can assign an editor for each node, when the AddNode event occurs. Use the Editor property to assign a predefined editor to a node. Use the SaveXML method to save the control's content to a specified location. The AllowDuplicateEntries property returns or sets a value that specifies whether the control supports nodes with the same key (duplicates).

The Name property indicates the name of the XML node being loaded. The Value property indicates the value of the XML node being loaded. The Image property of the Node object indicates the type of the XML node being loaded. The Image property holds the type of the XML node, like listed bellow:

- **NODE_ELEMENT** (1) The node represents an element (its nodeTypeString property is "element"). An Element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection, and EntityReference. The Element node can be the child of the Document, DocumentFragment, EntityReference, and Element nodes.
- **NODE_ATTRIBUTE** (2) The node represents an attribute of an element (its nodeTypeString property is "attribute"). An Attribute node can have the following child node types: Text and EntityReference. The Attribute node does not appear as the child

node of any other node type; it is not considered a child node of an Element.

- **NODE_TEXT** (3) The node represents the text content of a tag (its nodeTypeString property is "text"). A Text node cannot have any child nodes. The Text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.
- **NODE_CDATA_SECTION** (4) The node represents a CDATA section in the XML source (its nodeTypeString property is "cdatasection"). CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATASection node cannot have any child nodes. The CDATASection node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.
- **NODE_ENTITY_REFERENCE** (5) The node represents a reference to an entity in the XML document (its nodeTypeString property is "entityreference"). This applies to all entities, including character entity references. An EntityReference node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The EntityReference node can appear as the child of the Attribute, DocumentFragment, Element, and EntityReference nodes.
- **NODE_ENTITY** (6) The node represents an expanded entity (its nodeTypeString property is "entity"). An Entity node can have child nodes that represent the expanded entity (for example, Text and EntityReference nodes). The Entity node can appear as the child of the DocumentType node.
- **NODE_PROCESSING_INSTRUCTION** (7) The node represents a processing instruction from the XML document (its nodeTypeString property is "processinginstruction"). A ProcessingInstruction node cannot have any child nodes. The ProcessingInstruction node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
- **NODE_COMMENT** (8) The node represents a comment in the XML document (its nodeTypeString property is "comment"). A Comment node cannot have any child nodes. The Comment node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
- **NODE_DOCUMENT** (9) The node represents a document object, that as the root of the document tree, provides access to the entire XML document (its nodeTypeString property is "document"). It is created using the progID "Microsoft.XMLDOM" or through a data island using <XML> or <SCRIPT LANGUAGE=XML>. A Document node can have the following child node types: Element (maximum of one), ProcessingInstruction, Comment, and DocumentType. The Document node cannot appear as the child of any node types.
- **NODE_DOCUMENT_TYPE** (10) The node represents the document type declaration, indicated by the <!DOCTYPE> tag (its nodeTypeString property is "documenttype"). A DocumentType node can have the following child node types: Notation and Entity. The DocumentType node can appear as the child of the Document node.
- **NODE_DOCUMENT_FRAGMENT** (11) The node represents a document fragment (its nodeTypeString property is "documentfragment"). The DocumentFragment node associates a node or subtree with a document without actually being contained within

the document. A DocumentFragment node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The DocumentFragment node cannot appear as the child of any node types.

- **NODE_NOTATION** (12) The node represents a notation in the document type declaration (its nodeTypeString property is "notation"). A Notation node cannot have any child nodes. The Notation node can appear as the child of the DocumentType node.

Use the Images method to add images to the control, so each type of element in your XML document, has a graphic representation. So, the first icon in the Images collection indicates the NODE_ELEMENT type, the second icon in the Images collection indicates the NODE_ATTRIBUTE type, and so on.

property XMLGrid.MoveCursorOnCollapse as Boolean

Moves the cursor when a node is collapsed using the mouse.

Type	Description
Boolean	A boolean expression that indicates whether the control moves the cursor when user collapses a node.

By default, the MoveCursorOnCollapse property is True.

property XMLGrid.NodeByPosition (Position as Long) as Node

Retrieves a node giving its position.

Type	Description
Position as Long	A long expression that indicates the position of the node being requested.
Node	A Node object that indicates the node at position.

Use the NodeByPosition property to get a node giving its position. Use the Position property to change the node's position in the list of node's child nodes collection. Use the Visible property to hide a node. Use the NodeFromPoint property to get the node from cursor. The VisibleNodeCount property specifies the number of visible nodes.

The following sample displays the list of visible nodes as they are displayed:

```
With XMLGrid1
   Dim n As EXMLGRIDLibCtl.Node, i As Long
   i = 0
   Set n = .NodeByPosition(i)
   While Not n Is Nothing
      Debug.Print n.Name
      i = i + 1
      Set n = .NodeByPosition(i)
   Wend
End With
```

property XMLGrid.NodeFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as Node

Retrieves the node's 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 X location of the mouse pointer. The x values is always expressed in client coordinates
Node	A Node object where the point is.

Use the NodeFromPoint property to get the node from the cursor. Call the HitTest method to determine the location of the specified point relative to the client area of a xml grid view control.

The following VB sample prints the name of the node over the cursor:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With XMLGrid1
    Dim n As EXMLGRIDLibCtl.Node
    Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not n Is Nothing Then
      Debug.Print "Hovers '" & n.Name & "'."
    End If
  End With
End Sub
```

The following VB sample displays the hit test code while user moves the mouse:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With XMLGrid1
    Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
    h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
    If Not h = 0 Then
      If (Not n Is Nothing) Then
```

```
            Debug.Print "Node = " & n.Name & " H = " & Hex(h)
         Else
            Debug.Print "H = " & Hex(h)
         End If
      End If
   End With
End Sub
```

The following C++ sample prints the name of the node from the cursor:

```
#include "Node.h"
void OnMouseMoveXmlgrid1(short Button, short Shift, long X, long Y)
{
   CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
   if ( node.m_lpDispatch != NULL )
   {
      CString strName = node.GetName();
      OutputDebugString( strName );
   }
}
```

The following VB.NET sample prints the name of the node from the cursor:

```
Private Sub AxXMLGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent) Handles
AxXMLGrid1.MouseMoveEvent
   With AxXMLGrid1
      Dim n As EXMLGRIDLib.Node = .get_NodeFromPoint(e.x, e.y)
      If Not n Is Nothing Then
         Debug.Print("You have clicked the '" & n.Name & "'.")
      End If
   End With
End Sub
```

The following C# sample prints the name of the node from the cursor:

```
private void axXMLGrid1_MouseMoveEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent e)
{
```

```
    EXMLGRIDLib.Node node = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
    if (node != null)
        System.Diagnostics.Debug.Write(node.Name);
}
```

The following VFP sample prints the name of the node from the cursor:

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

with thisform.XMLGrid1
    n = .NodeFromPoint(x, y )
    if ( !isnull(n) )
        wait window nowait n.Name
    endif
endwith
```

property XMLGrid.NodeHeight as Long

Sets or gets a value that indicates the node's height.

Type	Description
Long	A long expression that indicates the height of nodes in the control, in pixels.

Use the NodeHeight property to indicate the height of the nodes in the control. By default, the NodeHeight property is 21 pixels. Use the Visible property to hide a node. Use the Remove method to remove a node.

property XMLGrid.Nodes as Nodes

Retrieves the Nodes collection.

Type	Description
Nodes	A Nodes object that holds the control's nodes collection.

Use the Nodes property to access the control's nodes collection. Use the Add method to add new nodes to the control. Use the Editors property to access the control's editors collection. Use the Editor property to assign an editor to a node. Use the Nodes property to access the node's child nodes collection. Use the ItemByPosition property to retrieve a node giving its position. Use the FirstNode property to retrieves the first child node, and the NextNode property to retrieve the next child node.

The following VB sample enumerates the nodes in the control (including the child nodes):

```
Private Sub enumerate(ByVal x As EXMLGRIDLibCtl.XMLGrid)
   With x.Nodes
      Dim i As Long
      For i = 0 To .Count - 1
         enumNodes .ItemByPosition(i)
      Next
   End With
End Sub

Private Sub enumNodes(ByVal n As EXMLGRIDLibCtl.node)
   Dim c As EXMLGRIDLibCtl.node
   Debug.Print n.Name
   Set c = n.FirstNode
   While Not c Is Nothing
      enumNodes c
      Set c = c.NextNode
   Wend
End Sub
```

The **enumerate** function enumerates the root nodes in the control. The **enumNodes** function enumerates recursively the child nodes for each node.

method XMLGrid.OLEDrag ()

Causes a component to initiate an OLE drag/drop operation.

Type	Description

Only for internal use.

property XMLGrid.OLEDropMode as exOLEDropModeEnum

Returns or sets how a target component handles drop operations

Type	Description
exOLEDropModeEnum	An exOLEDropModeEnum expression that indicates the OLE Drag and Drop mode.

The eXMLGrid component supports manual or automatic OLE Drag and Drop operation. See the OLEStartDrag and OLEDragDrop events for more details about implementing OLE drag and drop operations in the eXMLGrid component.

property XMLGrid.Picture as IPictureDisp

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

Type	Description
IPictureDisp	A Picture object that indicates the control's picture.

Use the Picture property to load a picture on the control's background. Use the PictureDisplay property to arrange the picture on the control's background. Use the Picture property to assign a picture to a node. Use the Images method to load a list of icons to the control. Use the Image property to assign an icon to a node.

property XMLGrid.PictureDisplay as PictureDisplayEnum

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

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

Use the PictureDisplay property to arrange how the control's picture is displayed on its background. Use the Picture property to load a picture into the control's background. Use the Picture property to assign a picture to a node. Use the Images method to load a list of icons to the control. Use the Image property to assign an icon to a node.

property XMLGrid.ReadOnly as Boolean

Specifies whether the control is read only.

Type	Description
Boolean	A boolean expression that indicates whether the control is read only.

Use the ReadOnly property to make the control read only. Use the Enabled property to disable the control. Use the Locked property to lock an editor. If the control is read only, the Edit event is never fired

method XMLGrid.Refresh ()

Refreshes the control.

Type	Description

Use the Refresh method to refresh the control's content. Use the BeginUpdate and EndUpdate methods to maintain performance while adding or changing multiple nodes.

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

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

Type	Description
Icon as Variant	A long expression that indicates the icon's handle
Index as Variant	A long expression that indicates the index where icon is inserted
Return	**Description**
Long	A long expression that indicates the index of the icon in the images collection

Use the ReplaceIcon property to add, remove or replace an icon in the control's images collection. Also, the ReplaceIcon property can clear the images collection. Use the Images method to attach an image list to the control.

The following sample shows how to add a new icon to control's images list:

i = XMLGrid1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle), in this case the i specifies the index where the icon was added

The following sample shows how to replace an icon into control's images list::

i = XMLGrid1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle, 0), in this case the i is zero, because the first icon was replaced.

The following sample shows how to remove an icon from control's images list:

XMLGrid1.ReplaceIcon 0, i, in this case the i must be the index of the icon that follows to be removed

The following sample shows how to clear the control's icons collection:

XMLGrid1.ReplaceIcon 0, -1

method XMLGrid.ResizeToFit ([Level as Variant], [IncludeNextLevels as Variant])

Resizes the control's level (and the next ones) so its content its fully visible.

Type	Description
Level as Variant	A Long expression that specifies the level to start resizing to fit.
IncludeNextLevels as Variant	A Boolean expression that specifies whether next levels are adjusted as well. If IncludeNextLevels parameter is missing, no next levels are included in the ResizeToFit method.

The ResizeToFit method resizes the control's level (and the next ones) so its content its fully visible. The user can [SHIFT +]double click the resizing level, so it gets fit. Use the LevelWidth property to specify the level's width. The control fires the ResizeLevel event when user resizes a level. Use the Level property to get the node's level.

method XMLGrid.SaveXML (Destination as Variant)

Saves the control's content as XML document to the specified location, using the MSXML parser.

Type	Description

Destination as Variant	This object can represent a file name, reference to a string member, an XML document object, or a custom object that supports persistence as follows:
	
	• 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:
	
	`XMLGrid1.SaveXML("sample.xml")`
	
	• 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:
	
	`Dim s As String`
	`XMLGrid1.SaveXML s`
	
	In VB.NET for /NET assembly, you should call such as :
	
	`Dim s As String = String.Empty`
	`Exmlgrid1.SaveXML(s)`
	
	In C# for /NET assembly, you should call such as :
	
	`string s = string.Empty;`
	`exmlgrid1.SaveXML(ref s);`
	
	• XML Document Object. For example:
	
	`Dim xmldoc as Object`
	`Set xmldoc = CreateObject("MSXML.DOMDocument")`
	`XMLGrid1.SaveXML(xmldoc)`
	
	• Custom object supporting persistence - Any other custom COM object that supports **QueryInterface** for **IStream**, **IPersistStream**, or **IPersistStreamInit** can also be provided here and the document will be saved accordingly. In the **IStream** case, the **IStream::Write**

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 Boolen expression that specifies whether saving the XML document was ok.

The SaveXML method saves control's content in XML format. Use the LoadXML method to load XML documents. The Name property indicates the name of the XML node being saved. The Value property indicates the value of the XML node being saved. The Image property of the Node object indicates the type of the XML node being saved. The Image property holds the type of the XML node, like listed bellow:

- **NODE_ELEMENT** (1) The node represents an element (its nodeTypeString property is "element"). An Element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection, and EntityReference. The Element node can be the child of the Document, DocumentFragment, EntityReference, and Element nodes.
- **NODE_ATTRIBUTE** (2) The node represents an attribute of an element (its nodeTypeString property is "attribute"). An Attribute node can have the following child node types: Text and EntityReference. The Attribute node does not appear as the child node of any other node type; it is not considered a child node of an Element.
- **NODE_TEXT** (3) The node represents the text content of a tag (its nodeTypeString property is "text"). A Text node cannot have any child nodes. The Text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.
- **NODE_CDATA_SECTION** (4) The node represents a CDATA section in the XML source (its nodeTypeString property is "cdatasection"). CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATASection node cannot have any child nodes. The CDATASection node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.
- **NODE_ENTITY_REFERENCE** (5) The node represents a reference to an entity in the XML document (its nodeTypeString property is "entityreference"). This applies to all entities, including character entity references. An EntityReference node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The EntityReference node can appear as the child of the Attribute, DocumentFragment, Element, and EntityReference nodes.
- **NODE_ENTITY** (6) The node represents an expanded entity (its nodeTypeString property is "entity"). An Entity node can have child nodes that represent the expanded entity (for example, Text and EntityReference nodes). The Entity node can appear as

the child of the DocumentType node.

- **NODE_PROCESSING_INSTRUCTION** (7) The node represents a processing instruction from the XML document (its nodeTypeString property is "processinginstruction"). A ProcessingInstruction node cannot have any child nodes. The ProcessingInstruction node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
- **NODE_COMMENT** (8) The node represents a comment in the XML document (its nodeTypeString property is "comment"). A Comment node cannot have any child nodes. The Comment node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
- **NODE_DOCUMENT** (9) The node represents a document object, that as the root of the document tree, provides access to the entire XML document (its nodeTypeString property is "document"). It is created using the progID "Microsoft.XMLDOM" or through a data island using <XML> or <SCRIPT LANGUAGE=XML>. A Document node can have the following child node types: Element (maximum of one), ProcessingInstruction, Comment, and DocumentType. The Document node cannot appear as the child of any node types.
- **NODE_DOCUMENT_TYPE** (10) The node represents the document type declaration, indicated by the <!DOCTYPE> tag (its nodeTypeString property is "documenttype"). A DocumentType node can have the following child node types: Notation and Entity. The DocumentType node can appear as the child of the Document node.
- **NODE_DOCUMENT_FRAGMENT** (11) The node represents a document fragment (its nodeTypeString property is "documentfragment"). The DocumentFragment node associates a node or subtree with a document without actually being contained within the document. A DocumentFragment node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The DocumentFragment node cannot appear as the child of any node types.
- **NODE_NOTATION** (12) The node represents a notation in the document type declaration (its nodeTypeString property is "notation"). A Notation node cannot have any child nodes. The Notation node can appear as the child of the DocumentType node.

The Destination's type can be one of the following:

xmlDestination	Description
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: `XMLGrid11.SaveXML("sample.xml")`

XML Document Object	For example:
	`Dim xmldoc as Object`
	`Set xmldoc = CreateObject("MSXML.DOMDocument")`
	`XMLGrid11.SaveXML(xmldoc)`

Custom object supporting persistence	Any other custom COM object that supports **QueryInterface** for **IStream**, **IPersistStream**, or **IPersistStreamInit** can also be provided here and the document will be saved accordingly. In the **IStream** case, the **IStream::Write** 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.

method XMLGrid.Scroll (Type as ScrollEnum, [ScrollTo as Variant])

Scrolls the control's content.

Type	Description
Type as ScrollEnum	A ScrollEnum expression that indicates type of scrolling being performed.
ScrollTo as Variant	A long expression that indicates the position where the control is scrolled when Type is exScrollVTo or exScrollHTo. If the ScrollTo parameter is missing, 0 value is used.

Use the Scroll method to scroll the control's content by code. Use the Scrollbars property specifies which scroll bars will be visible on the control. Use the EnsureVisibleNode method to ensure that a specified node fits the control's client area.

property XMLGrid.ScrollBars as ScrollBarsEnum

Specifies the type of scroll bars that control has.

Type	Description
ScrollBarsEnum	A ScrollBarsEnum expression that indicates which scroll bars will be visible in the control.

Use the ScrollBars property to disable the control's scroll bars. The ScrollPos property specifies the vertical/horizontal scroll position. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar.

property XMLGrid.ScrollButtonHeight as Long

Specifies the height of the button in the vertical scrollbar.

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

By default, the ScrollButtonHeight property is -1. If the ScrollButtonHeight property is -1, the control uses the default height (from the system) for the buttons in the vertical scroll bar. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property XMLGrid.ScrollButtonWidth as Long

Specifies the width of the button in the horizontal scrollbar.

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

By default, the ScrollButtonWidth property is -1. If the ScrollButtonWidth property is -1, the control uses the default width (from the system) for the buttons in the horizontal scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property XMLGrid.ScrollFont (ScrollBar as ScrollBarEnum) as IFontDisp

Retrieves or sets the scrollbar's font.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical or the horizontal scroll bar.
IFontDisp	A Font object

Use the ScrollFont property to specify the font in the control's scroll bar. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar.

property XMLGrid.ScrollHeight as Long

Specifies the height of the horizontal scrollbar.

Type	Description
Long	A long expression that defines the height of the horizontal scroll bar.

By default, the ScrollHeight property is -1. If the ScrollHeight property is -1, the control uses the default height of the horizontal scroll bar from the system. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property XMLGrid.ScrollOrderParts(ScrollBar as ScrollBarEnum) as String

Specifies the order of the buttons in the scroll bar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the order of buttons is displayed.
String	A String expression that indicates the order of the buttons in the scroll bar. The list includes expressions like l, l1, ..., l5, t, r, r1, ..., r6 separated by comma, each expression indicating a part of the scroll bar, and its position indicating the displaying order.

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

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

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

The list of supported literals in the ScrollOrderParts property is:

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

- **r1** for exRightB1Part, (R1) The first additional button in the right or down side.
- **r2** for exRightB2Part, (R2) The second additional button in the right or down side.
- **r3** for exRightB3Part, (R3) The third additional button in the right or down side.
- **r4** for exRightB4Part, (R4) The forth additional button in the right or down side.
- **r5** for exRightB5Part, (R5) The fifth additional button in the right or down side.
- **r6** for exRightB6Part, (R6) The sixth additional button in the right or down side.

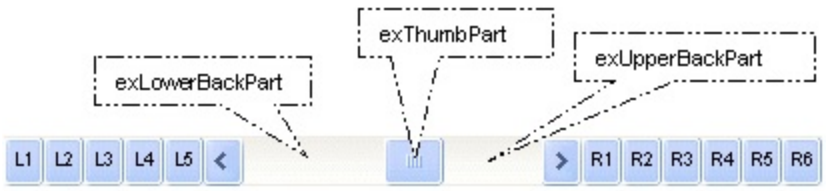
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property XMLGrid.ScrollPartCaption(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as String

Specifies the caption being displayed on the specified scroll part.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the caption is displayed.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll where the text is displated
String	A String expression that specifies the caption being displayed on the part of the scroll bar.

Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar. Use the ScrollFont property to specify the font in the control's scroll bar. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.



By default, the following parts are shown:

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

The following VB sample adds up and down additional buttons to the control's vertical scroll bar :

```
With XMLGrid1
  .BeginUpdate
```

```
    .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
    .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
    .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
  .EndUpdate
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxXMLGrid1
  .BeginUpdate()
  .set_ScrollPartVisible(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part Or
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, True)
  .set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
  .set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2")
  .EndUpdate()
End With
```

The following C# sample adds up and down additional buttons to the control's vertical scroll bar :

```
axXMLGrid1.BeginUpdate();
axXMLGrid1.set_ScrollPartVisible(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part |
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, true);
axXMLGrid1.set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axXMLGrid1.set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axXMLGrid1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_xmlGrid.BeginUpdate();
m_xmlGrid.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
```

```
/*exRightB1Part*/, TRUE );
m_xmlGrid.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_xmlGrid.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img>
</img>2") );
m_xmlGrid.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.XMLGrid1
   .BeginUpdate
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWith
```

```
*** ActiveX Control Event ***
LPARAMETERS scrollpart

wait window nowait ltrim(str(scrollpart))
```

property XMLGrid.ScrollPartCaptionAlignment(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as AlignmentEnum

Specifies the alignment of the caption in the part of the scroll bar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the caption is displayed
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll where the text is displayed
AlignmentEnum	An AlignmentEnum expression that specifies the alignment of the caption in the part of the scrollbar

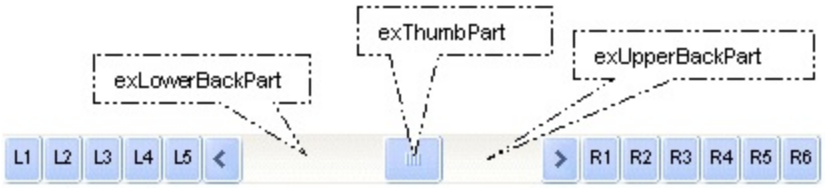
The ScrollPartCaptionAlignment property specifies the alignment of the caption in the part of the scroll bar. By default, the caption is centered. Use the ScrolPartCaption property to specify the caption being displayed on specified part of the scroll bar. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar.

property XMLGrid.ScrollPartEnable(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

Indicates whether the specified scroll part is enabled or disabled.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the part is enabled or disabled.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll bar being enabled or disabled.
Boolean	A Boolean expression that specifies whether the scrollbar's part is enabled or disabled.

By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.

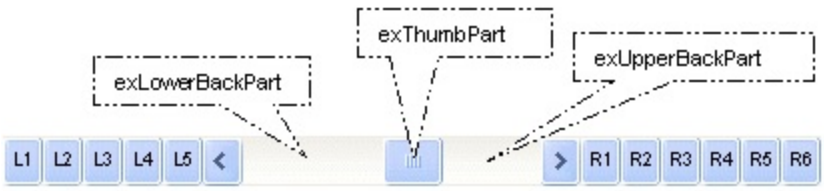


property XMLGrid.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

Indicates whether the specified scroll part is visible or hidden.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the part is visible or hidden.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll bar being visible
Boolean	A Boolean expression that specifies whether the scrollbar's part is visible or hidden.

Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar. Use the Background property to change the visual appearance for any part in the control's scroll bar. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.



By default, the following parts are shown:

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

The following VB sample adds up and down additional buttons to the control's vertical scroll bar :

```
With XMLGrid1
  .BeginUpdate
```

```
    .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
    .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
    .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
  .EndUpdate
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxXMLGrid1
  .BeginUpdate()
  .set_ScrollPartVisible(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part Or
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, True)
  .set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
  .set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2")
  .EndUpdate()
End With
```

The following C# sample adds up and down additional buttons to the control's vertical scroll bar :

```
axXMLGrid1.BeginUpdate();
axXMLGrid1.set_ScrollPartVisible(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part |
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, true);
axXMLGrid1.set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axXMLGrid1.set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axXMLGrid1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_xmlGrid.BeginUpdate();
m_xmlGrid.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
```

```
/*exRightB1Part*/, TRUE );
m_xmlGrid.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_xmlGrid.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img>
</img>2") );
m_xmlGrid.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.XMLGrid1
   .BeginUpdate
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWith
```

```
*** ActiveX Control Event ***
LPARAMETERS scrollpart

wait window nowait ltrim(str(scrollpart))
```

property XMLGrid.ScrollPos(Vertical as Boolean) as Long

Specifies the vertical/horizontal scroll position.

Type	Description
Vertical as Boolean	A Boolean expression that specifies the Vertical (True) or Horizontal (False) scroll bar of the control.
Long	A Long expression that indicates the control's scroll bar position.

The ScrollPos property specifies the vertical/horizontal scroll position. Use the ScrollPos property to determine or change the control's scroll position. The Layout property can be used to save and restore the control's layout which includes: scrolling position, selected nodes, expanded nodes, and so on. The ScrollBars property shows or hides the control's scroll bars. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar.

property XMLGrid.ScrollThumbSize(ScrollBar as ScrollBarEnum) as Long

Specifies the size of the thumb in the scrollbar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical or the horizontal scroll bar.
Long	A long expression that defines the size of the scrollbar's thumb.

Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb. By default, the ScrollThumbSize property is -1, that makes the control computes automatically the size of the thumb based on the scrollbar's range. If case, use the fixed size for your thumb when you change its visual appearance using the Background(exVSThumb) or Background(exHSThumb) property. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar.

property XMLGrid.ScrollToolTip(ScrollBar as ScrollBarEnum) as String

Specifies the tooltip being shown when the user moves the scroll box.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical scroll bar or the horizontal scroll bar.
String	A string expression being shown when the user clicks and moves the scrollbar's thumb.

Use the ScrollToolTip property to specify whether the control displays a tooltip when the user clicks and moves the scrollbar's thumb. By default, the ScrollToolTip property is empty. If the ScrollToolTip property is empty, the tooltip is not shown when the user clicks and moves the thumb of the scroll bar. Use the SortPartVisible property to specify the parts being visible in the control's scroll bar.

property XMLGrid.ScrollWidth as Long

Specifies the width of the vertical scrollbar.

Type	Description
Long	A long expression that defines the width of the vertical scroll bar.

By default, the ScrollWidth property is -1. If the ScrollWidth property is -1, the control uses the default width of the vertical scroll bar from the system. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property XMLGrid.Search (What as String, [How as Variant]) as Node

Searches for a node.

Type	Description
What as String	A String being searched.
How as Variant	An AutoSearchEnum expression that defines the way the control searches for the What string. If How parameter is missing, the control's AutoSearch property indicates the way the control searches for the What string.
Node	A Node object being found or null if nothing is found.

The Search property searches programmatically for for a node. The AutoSearch property defines how the control's 'incremental search' works. The Search method starts searching for specific node from the current focused node. The FocusNode property specifies the control's focused node. If no result is found, the Search property returns a null object.

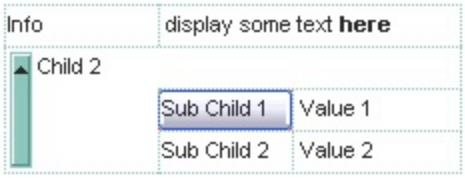
property XMLGrid.SelBackColor as Color

Specifies the selection's background color.

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

Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The SelBackColorCollapse property specifies the selection's background color, when the node is collapsed. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. The SelBackMode property specifies the way the control displays the selected nodes.

How do I assign a new look for the selected item?



The following VB sample changes the visual appearance for the selected node. The SelBackColor property indicates the selection background color. Shortly, we need to add a skin to the Appearance object using the Add method, and we need to set the last 7 bits in the SelBackColor property to indicates the index of the skin that we want to use. The sample applies the "  " to the selected node:

```
With XMLGrid1
  With .VisualAppearance
    .Add &H22, App.Path + "\selected.ebn"
  End With
  .SelForeColor = RGB(0, 0, 0)
  .SelBackColor = .SelBackColor Or &H22000000
  .SelBackColorChild = .BackColor
  .SelForeColorChild = vbBlack
```

```
End With
```

The following C++ sample changes the visual appearance for the selected node:

```
#include "Appearance.h"
m_xmlgrid.GetVisualAppearance().Add( 0x22,
COleVariant(_T("D:\\Temp\\EXMLGrid.Help\\selected.ebn")) );
m_xmlgrid.SetSelBackColor( RGB(0,0,255) | 0x22000000 );
m_xmlgrid.SetSelForeColor( 0 );
m_xmlgrid.SetSelBackColorChild(m_xmlgrid.GetBackColor());
m_xmlgrid.SetSelForeColorChild( 0 );
```

The following VB.NET sample changes the visual appearance for the selected node:

```
With AxXMLGrid1
   With .VisualAppearance
      .Add(&H22, "D:\Temp\EXMLGrid.Help\selected.ebn")
   End With
   .SelForeColor = Color.Black
   .Template = "SelBackColor = 587137024"
   .SelBackColorChild = .BackColor
   .SelForeColorChild = Color.Black
End With
```

where the 587137024 value is the hexa representation of 0x22FF0000

The following C# sample changes the visual appearance for the selected node:

```
axXMLGrid1.VisualAppearance.Add(0x22, "d:\\temp\\EXMLGrid.Help\\selected.ebn");
axXMLGrid1.Template = "SelBackColor = 587137024";
axXMLGrid1.SelForeColorChild = Color.Black;
axXMLGrid1.SelBackColorChild = axXMLGrid1.BackColor;
```

where the 587137024 value is the hexa representation of 0x22FF0000.

The following VFP sample changes the visual appearance for the selected node:

```
With thisform.XMLGrid1
   With .VisualAppearance
      .Add(34, "D:\Temp\EXMLGrid.Help\selected.ebn")
```

```
    EndWith
    .SelForeColor = RGB(0, 0, 0)
    .SelBackColor = RGB(0,0,255) + 570425344
    .SelBackColorChild = .BackColor
    .SelForeColorChild = RGB(0, 0, 0)
EndWith
```

How do I assign a new look for the selected item?

The component supports skinning parts of the control, including the selected item. Shortly, the idea is that identifier of the skin being added to the Appearance collection is stored in the first significant byte of property of the color type. In our case, we know that the SelBackColor property changes the background color for the selected item. This is what we need to change. In other words, we need to change the visual appearance for the selected item, and that means changing the background color of the selected item. So, the following code (blue code) changes the appearance for the selected item:

```
With XMLGrid1
    .VisualAppearance.Add &H34, App.Path + "\aqua.ebn"
    .SelBackColor = &H34000000
End With
```

Please notice that the 34 hexa value is arbitrary chosen, it is not a predefined value. Shortly, we have added a skin with the identifier 34, and we specified that the SelBackColor property should use that skin, in order to change the visual appearance for the selected item. Also, please notice that the 34 value is stored in the first significant byte, not in other position. For instance, the following sample doesn't use any skin when displaying the selected item:

```
With XMLGrid1
    .VisualAppearance.Add &H34, App.Path + "\aqua.ebn"
    .SelBackColor = &H34
End With
```

This code (red code) DOESN'T use any skin, because the 34 value is not stored in the higher byte of the color value. The sample just changes the background color for the selected item to some black color (RGB(0,0,34)). So, please pay attention when you want to use a skin and when to use a color. Simple, if you are calling &H**34**000000, you have 34 followed by 6 (six) zeros, and that means the first significant byte of the color expression. Now, back to the problem. The next step is how we are creating skins? or EBN files? The Exontrol's exbutton component includes a builder tool that saves skins to EBN

files. So, if you want to create new skin files, you need to download and install the exbutton component from our web site. Once that the exbutton component is installed, please follow the steps.

Let's say that we have a BMP file, that we want to stretch on the selected item's background.

1. Open the VB\Builder or VC\Builder sample
2. Click the **New File** button (on the left side in the toolbar), an empty skin is created.
3. Locate the **Background** tool window and select the **Picture\Add New** item in the menu, the Open file dialog is opened.
4. Select the picture file (GIF, BMP, JPG, JPEG). You will notice that the visual appearance of the focused object in the skin is changed, actually the picture you have selected is tiled on the object's background.
5. Select the **None** item, in the Background tool window, so the focused object in the skin is not displaying anymore the picture being added.
6. Select the **Root** item in the skin builder window (in the left side you can find the hierarchy of the objects that composes the skin), so the Root item is selected, and so focused.
7. Select the picture file you have added at the step 4, so the Root object is filled with the picture you have chosen.
8. Resize the picture in the Background tool window, until you reach the view you want to have, no black area, or change the CX and CY fields in the Background tool window, so no black area is displayed.
9. Select **Stretch** button in the Background tool window, so the Root object stretches the picture you have selected.
10. Click the **Save a file** button, and select a name for the new skin, click the Save button after you typed the name of the skin file. Add the .ebn extension.
11. Close the builder

You can always open the skin with the builder and change it later, in case you want to change it.

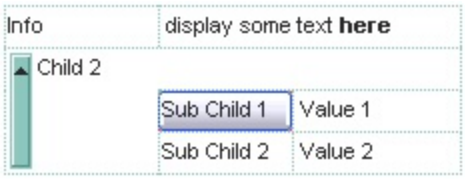
Now, create a new project, and insert the component where you want to use the skin, and add the skin file to the Appearance collection of the object, using blue code, by changing the name of the file or the path where you have selected the skin. Once that you have added the skin file to the Appearance collection, you can change the visual appearance for parts of the controls that supports skinning. **Usually the properties that changes the background color for a part of the control supports skinning as well.**

property XMLGrid.SelBackColorChild as Color

Specifies the selection's background color on the value section.

Type	Description
Color	A color expression that indicates the background color for child nodes of the selected nodes. 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.

Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The SelBackColorCollapse property specifies the selection's background color, when the node is collapsed. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. The SelBackMode property specifies the way the control displays the selected nodes.



The following VB sample changes the visual appearance for the selected node. The SelBackColor property indicates the selection background color. Shortly, we need to add a skin to the Appearance object using the Add method, and we need to set the last 7 bits in the SelBackColor property to indicates the index of the skin that we want to use. The sample applies the "  " to the selected node:

```
With XMLGrid1
   With .VisualAppearance
      .Add &H22, App.Path + "\selected.ebn"
   End With
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = .SelBackColor Or &H22000000
   .SelBackColorChild = .BackColor
   .SelForeColorChild = vbBlack
End With
```

The following C++ sample changes the visual appearance for the selected node:

```
#include "Appearance.h"
m_xmlgrid.GetVisualAppearance().Add( 0x22,
COleVariant(_T("D:\\Temp\\EXMLGrid.Help\\selected.ebn")) );
m_xmlgrid.SetSelBackColor( RGB(0,0,255) | 0x22000000 );
m_xmlgrid.SetSelForeColor( 0 );
m_xmlgrid.SetSelBackColorChild(m_xmlgrid.GetBackColor());
m_xmlgrid.SetSelForeColorChild( 0 );
```

The following VB.NET sample changes the visual appearance for the selected node:

```
With AxXMLGrid1
   With .VisualAppearance
      .Add(&H22, "D:\Temp\EXMLGrid.Help\selected.ebn")
   End With
   .SelForeColor = Color.Black
   .Template = "SelBackColor = 587137024"
   .SelBackColorChild = .BackColor
   .SelForeColorChild = Color.Black
End With
```

where the 587137024 value is the hexa representation of 0x22FF0000

The following C# sample changes the visual appearance for the selected node:

```
axXMLGrid1.VisualAppearance.Add(0x22, "d:\\temp\\EXMLGrid.Help\\selected.ebn");
axXMLGrid1.Template = "SelBackColor = 587137024";
axXMLGrid1.SelForeColorChild = Color.Black;
axXMLGrid1.SelBackColorChild = axXMLGrid1.BackColor;
```

where the 587137024 value is the hexa representation of 0x22FF0000.

The following VFP sample changes the visual appearance for the selected node:

```
With thisform.XMLGrid1
   With .VisualAppearance
      .Add(34, "D:\Temp\EXMLGrid.Help\selected.ebn")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
```

```
    .SelBackColor = RGB(0,0,255) + 570425344
    .SelBackColorChild = .BackColor
    .SelForeColorChild = RGB(0, 0, 0)
EndWith
```

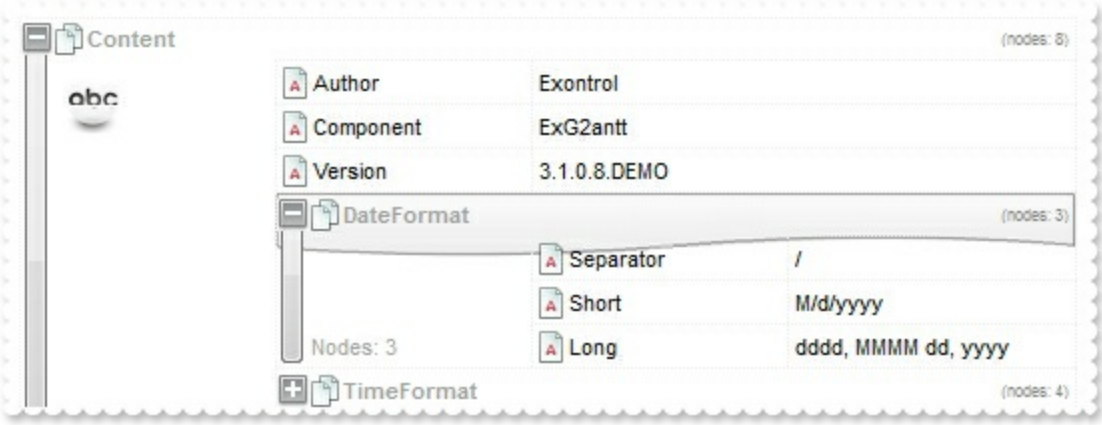
property XMLGrid.SelBackColorCollapse as Color

Specifies the selection's background color, when the node is collapsed.

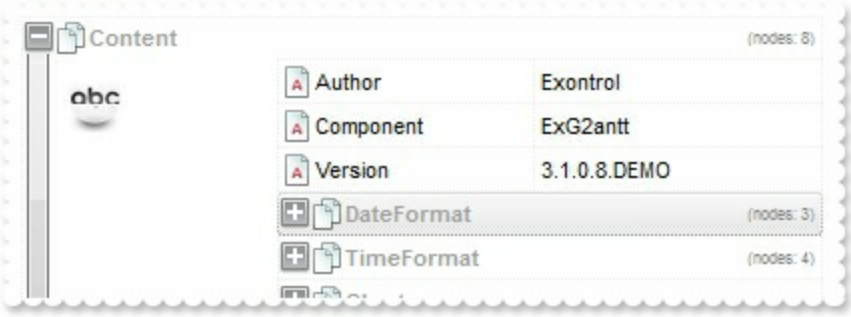
Type	Description
Color	A color expression that indicates the background color for child nodes of the selected nodes. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

By default, the SelBackColorCollapse property is -1, which indicates it has no effect. The SelBackColorCollapse property specifies the selection's background color, when the node is collapsed. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. The SelBackMode property specifies the way the control displays the selected nodes.

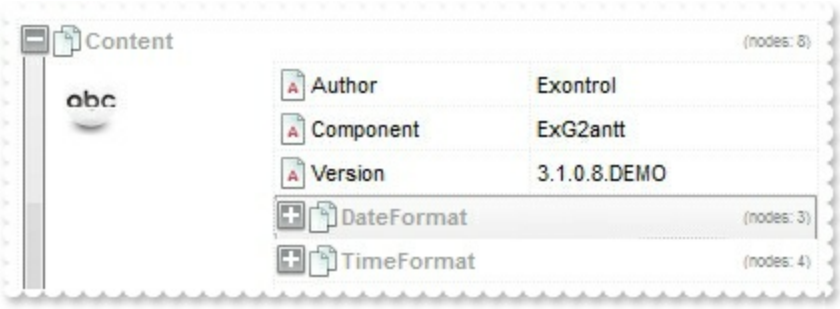
The following screen shot shows the "DateFormat" node when it expanded (SelBackColor, SelBackColorChild property is applied, using an EBN object)



The following screen shot shows the "DateFormat" node when it collapsed (SelBackColorCollapse is NOT -1, so it is applied, so the SelBackColor, SelBackColorChild property is NOT applied)



The following screen shot shows the "DateFormat" node when it collapsed (SelBackColorCollapse is -1, so it is NOT applied, so the SelBackColor, SelBackColorChild property is applied)



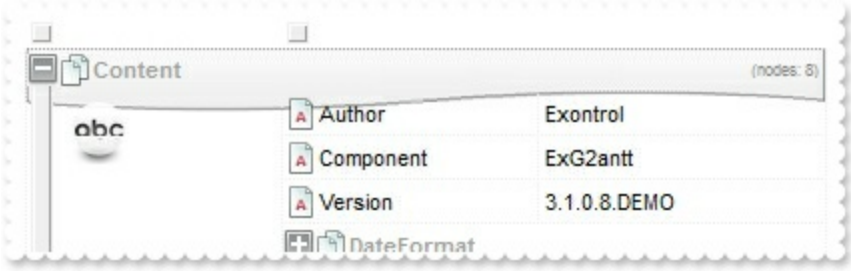
property XMLGrid.SelBackMode as BackModeEnum

Retrieves or sets a value that indicates whether the selection is transparent or opaque.

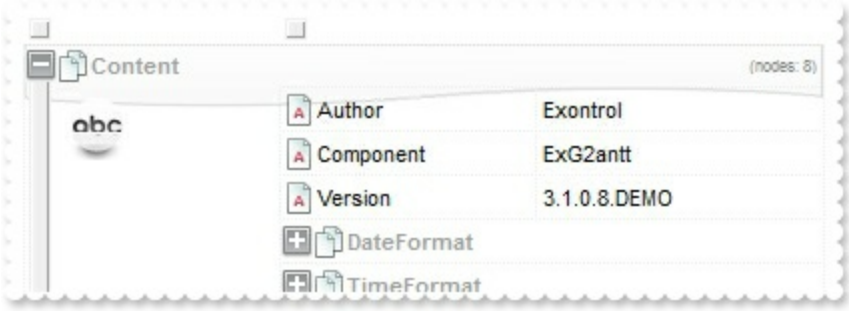
Type	Description
BackModeEnum	A BackModeEnum expression that specifies the way the control displays the selected nodes.

By default, the SelBackMode property is exOpaque. Use the SelBackMode property to display the selected nodes using a semi-transparent color. The SingleSel property specifies whether the control supports single or multiple nodes. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes.

The following screen shot shows the control, while SelBackMode property is exOpaque:



The following screen shot shows the control, while SelBackMode property is exTransparent:



method XMLGrid.SelectAll ()

Selects all nodes. The property is available only if the SingleSel property is False.

Type	Description

property XMLGrid.SelectCount as Long

Specifies the number of selected node.

Type	Description
Long	A long expression that indicates the number of selected nodes.

Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. Use the FocusNode property to retrieve the focused node. Use the SingleSel property to specify whether the control support single or multiple selection.

The following VB sample enumerates the selected node(s):

```
With XMLGrid1
   Dim i As Long
   For i = 0 To .SelectCount - 1
      Debug.Print .SelectedNode(i).Name
   Next
End With
```

The following C++ sample enumerates the selected node(s):

```
for ( long i = 0; i < m_xmlgrid.GetSelectCount(); i++ )
{
   CNode node = m_xmlgrid.GetSelectedNode( COleVariant( i ) );
   OutputDebugString( node.GetName() );
}
```

The following VB.NET sample enumerates the selected node(s):

```
With AxXMLGrid1
   Dim i As Long
   For i = 0 To .SelectCount - 1
      Debug.Write(.get_SelectedNode(i).Name())
   Next
End With
```

The following C# sample enumerates the selected node(s):

```
for (int i = 0; i < axXMLGrid1.SelectCount; i++)
{
   EXMLGRIDLib.Node node = axXMLGrid1.get_SelectedNode(i);
   System.Diagnostics.Debug.Write(node.Name);
}
```

The following VFP sample enumerates the selected node(s):

```
With thisform.XMLGrid1
   local i
   For i = 0 To .SelectCount - 1
      wait window nowait .SelectedNode(i).Name
   Next
EndWith
```

property XMLGrid.SelectedNode ([Index as Variant]) as Node

Retrieves the selected node.

Type	Description
Index as Variant	A long expression that indicates the index of selected node
Node	A Node object being requested.

Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the SelectCount property to get the number of selected nodes. Use the Selected property to select a node. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. Use the FocusNode property to retrieve the focused node. Use the SingleSel property to specify whether the control support single or multiple selection. The control fires the SelectionChanged event when user changes the selection.

The following VB sample enumerates the selected node(s):

```
With XMLGrid1
   Dim i As Long
   For i = 0 To .SelectCount - 1
      Debug.Print .SelectedNode(i).Name
   Next
End With
```

The following C++ sample enumerates the selected node(s):

```
for ( long i = 0; i < m_xmlgrid.GetSelectCount(); i++ )
{
   CNode node = m_xmlgrid.GetSelectedNode( COleVariant( i ) );
   OutputDebugString( node.GetName() );
}
```

The following VB.NET sample enumerates the selected node(s):

```
With AxXMLGrid1
   Dim i As Long
   For i = 0 To .SelectCount - 1
      Debug.Write(.get_SelectedNode(i).Name())
   Next
End With
```

The following C# sample enumerates the selected node(s):

```
for (int i = 0; i < axXMLGrid1.SelectCount; i++)
{
   EXMLGRIDLib.Node node = axXMLGrid1.get_SelectedNode(i);
   System.Diagnostics.Debug.Write(node.Name);
}
```

The following VFP sample enumerates the selected node(s):

```
With thisform.XMLGrid1
   local i
   For i = 0 To .SelectCount - 1
      wait window nowait .SelectedNode(i).Name
   Next
EndWith
```

property XMLGrid.SelForeColor as Color

Specifies the selection foreground's color.

Type	Description
Color	A color expression that indicates the foreground color of selected nodes.

Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. The SelBackMode property specifies the way the control displays the selected nodes.

property XMLGrid.SelForeColorChild as Color

Specifies the selection's background color on the value section.

Type	Description
Color	A color expression that indicates the foreground color for child nodes of the selected nodes.

Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The property has effect while its value is not -1. In other words, use the -1 to prevent apply the color on the node's background/foreground. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. The SelBackMode property specifies the way the control displays the selected nodes.

property XMLGrid.ShowFocusRect as Boolean

Retrieves or sets a value indicating whether the control draws a thin rectangle around the focused item.

Type	Description
Boolean	A boolean expression that specifies a value indicating whether the control draws a thin rectangle around the focused node.

Use the ShowFocusRect property to specify whether the control marks the focused node. Use the FocusNode property to get the control's focused node. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node.

property XMLGrid.ShowImageList as Boolean

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

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

The ShowImageList property has effect only at design time. Use the Images method to load a list of icons to the control.

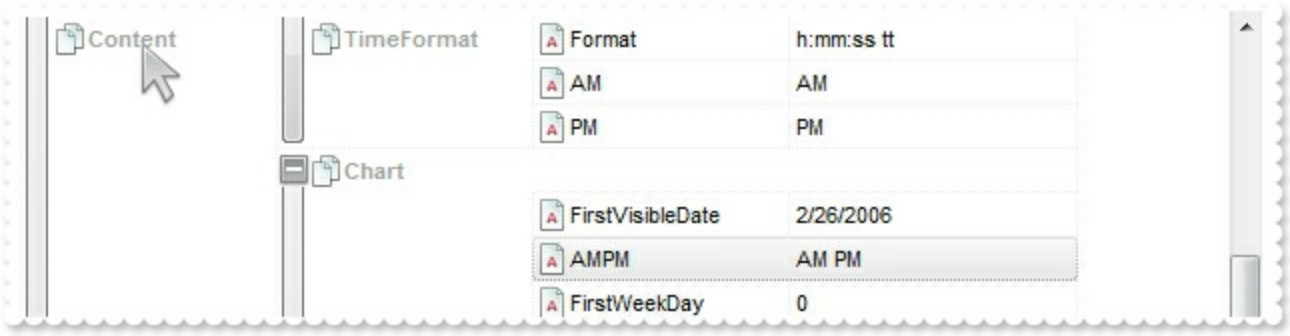
property XMLGrid.ShowPartialParent as ShowPartialParentEnum

Specifies where a partial-visible parent shows its content.

Type	Description
ShowPartialParentEnum	A ShowPartialParentEnum expression that specifies where a partial-visible parent shows its content.

By default, the ShowPartialParent property is exShowPartialParentTop, which indicates that the content of a partial-parent, is displayed on the top of the control (like you can see in the image bellow). Use the ShowPartialParent property to hide the partial parent content, or to display it on the focused node.

The following screen shot shows the content of the parent's node on the top of the control:



method XMLGrid.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	A String expression that indicates the description of the tooltip.
Title as Variant	If present, A String expression that indicates the title of the tooltip.
Alignment as Variant	A long expression that indicates the alignment of the tooltip relative to the position of the cursor. If missing, the tooltip is aligned to the left/top corder.
X as Variant	A single that specifies the current X location of the mouse pointer. The x values is always expressed in screen coordinates. If missing or -1, the current mouse X position is used. A string expression that indicates the offset to move the tooltip window relative to the cursor position.
Y as Variant	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in screen coordinates. If missing or -1, the current mouse Y position is used. A string expression that indicates the offset to move the tooltip window relative to the cursor position.

Use the ShowToolTip method to display 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 ShowToolTip method has no effect if the ToolTip and Title parameters are empty. The ToolTip/ToolTipTitle property assigns a tooltip to a node.

The Alignment parameter can be one of the followings:

- 0 - exTopLeft
- 1 - exTopRight
- 2 - exBottomLeft
- 3 - exBottomRight
- 0x10 - exCenter
- 0x11 - exCenterLeft
- 0x12 - exCenterRight

- 0x13 - exCenterTop
- 0x14 - exCenterBottom

Use numeric values as strings for X and Y parameters, to move the tooltip window relative to the position of the cursor. For instance, ShowToolTp("text",,,"11","12"), means that the tooltip window is moved 11 pixels on the X axis, and 12 pixels on the Y axis, before showing it in the default position. In this case the X and Y parameters MUST be passed as strings not as LONG values.

The following VB sample displays (as tooltip) the identifier of the anchor element from the cursor:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
   With XMLGrid1
      .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 AxXMLGrid1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As AxEXXMLGRIDLib._IXMLGridEvents_MouseMoveEvent) Handles AxXMLGrid1.MouseMoveEvent
   With AxXMLGrid1
      .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 axXMLGrid1_MouseMoveEvent(object sender, AxEXXMLGRIDLib._IXMLGridEvents_MouseMoveEvent e)
{
   axXMLGrid1.ShowToolTip(axXMLGrid1.get_AnchorFromPoint(-1, -1));
}
```

The following C++ sample displays (as tooltip) the identifier of the anchor element from the cursor:

```
void OnMouseMoveXMLGrid1(short Button, short Shift, long X, long Y)
{
   COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
   m_xmlGrid.ShowToolTip( m_xmlGrid.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 .XMLGrid1
      .ShowToolTip(.AnchorFromPoint(-1, -1))
   EndWith
endwith
```

property XMLGrid.SingleSel as Boolean

Specifies whether the control supports single or multiple selection.

Type	Description
Boolean	A boolean expression that indicates whether the control supports single or multiple selection.

Use the SingleSel property to allow multiple selection. By default, the SingleSel property it True. Use the HideSelection property to specify whether the selection is hidden when control loses the focus. Use the FocusNode property to retrieve the focused node. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes. The control fires the SelectionChanged event when user changes the selection. Use the Selected property to select a node. The SelBackMode property specifies the way the control displays the selected nodes.

The following sample displays the list of selected nodes:

```
With XMLGrid1
   Dim i As Long
   For i = 0 To .SelectCount - 1
      Debug.Print .SelectedNode(i).Name
   Next
End With
```

property XMLGrid.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 executes a template string and retrieves the result.

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

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

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

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

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

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

property XMLGrid.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.*

property XMLGrid.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 Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the ShowToolTip method to display a custom tooltip.

property XMLGrid.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

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

Use the ToolTipFont property to assign a font for the control's tooltip. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window.

property XMLGrid.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.

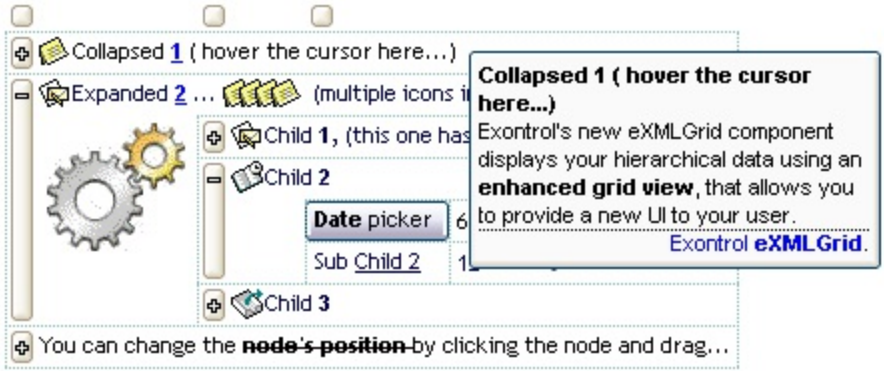
If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to assign a font for 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. Use the ShowToolTip method to display a custom tooltip.

property XMLGrid.ToolTipWidth as Long

Specifies a value that indicates the width of the tooltip window, in pixels.

Type	Description
Long	A long expression that indicates the width of the tooltip window.

Use the ToolTipWidth property to change the tooltip window width. The height of the tooltip window is automatically computed based on tooltip's description. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipFont property to assign a font for the control's tooltip. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the ShowToolTip method to display a custom tooltip.



method XMLGrid.UnselectAll ()

Unselects all nodes. The property is available only if the SingleSel property is False.

Type	Description

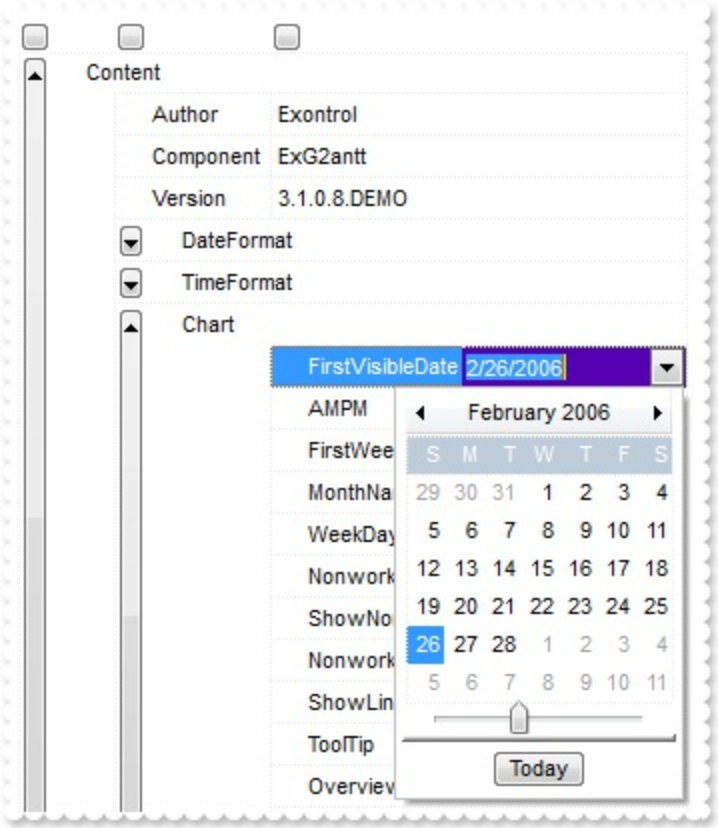
property XMLGrid.UseVisualTheme as UIVisualThemeEnum

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

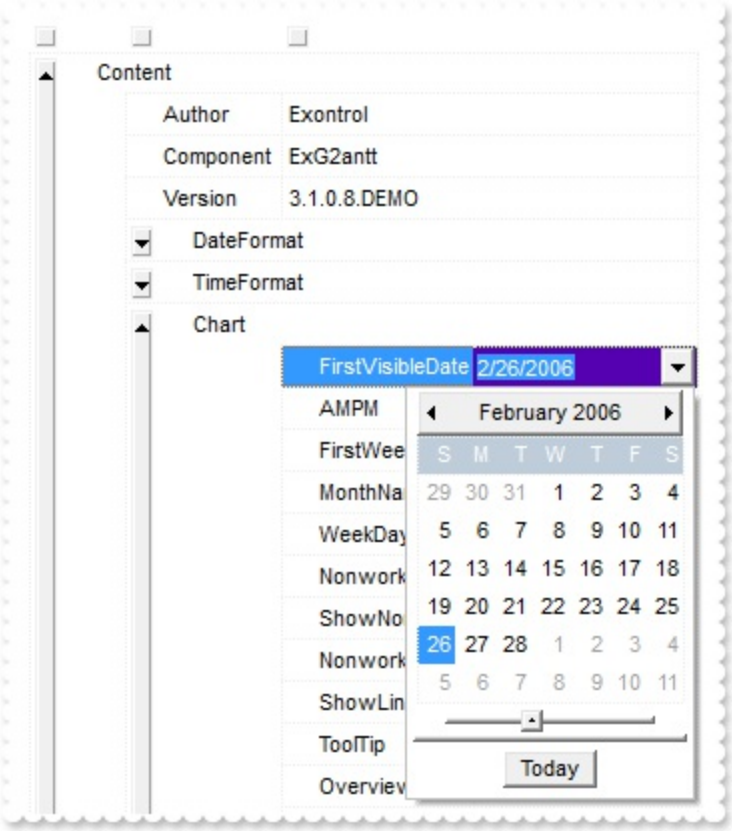
Type	Description
UIVisualThemeEnum	An UIVisualThemeEnum expression that specifies which UI parts of the control are shown using the current visual theme.

By default, the UseVisualTheme property is exDefaultVisualTheme, which means that all known UI parts are shown as in the current theme. The UseVisualTheme property may specify the UI parts that you need to enable or disable the current visual theme. The UI Parts are like header, filterbar, check-boxes, buttons and so on. The UseVisualTheme property has effect only a current theme is selected for your desktop. The UseVisualTheme property. Use the Appearance property of the control to provide your own visual appearance using the EBN files.

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



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


Content
Author
Exontrol
Component
ExG2antt
Version
3.1.0.8.DEMO
DateFormat
TimeFormat
Chart
FirstVisibleDate
2/26/2006
AMPM
FirstWee
MonthNa
WeekDay
Nonwork
ShowNo
Nonwork
ShowLin
ToolTip
Overviev
February 2006
S M T W T F S
29 30 31 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 1 2 3 4
5 6 7 8 9 10 11
Today

property XMLGrid.Version as String

Retrieves the control's version.

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

The version property specifies the control's version.

property XMLGrid.VisibleLevelCount as Long

Returns a value that indicates the number of visible levels in the tree control.

Type	Description
Long	A long expression that indicates the number of visible levels.

Use the VisibleLevelCount property to specify the number of levels being displayed. Use the LevelWidth property to specify the level's width. Use the Level property to get the node's level.

property XMLGrid.VisibleNodeCount as Long

Specifies the number of visible nodes.

Type	Description
Long	A long expression that indicates the number of visible nodes.

The VisibleNodeCount property specifies the number of visible nodes. Use the NodeByPosition property to access a node by its position. Use the Visible property to hide a node. Use the Expanded property to expand or collapse a node.

The following sample displays the list of visible nodes:

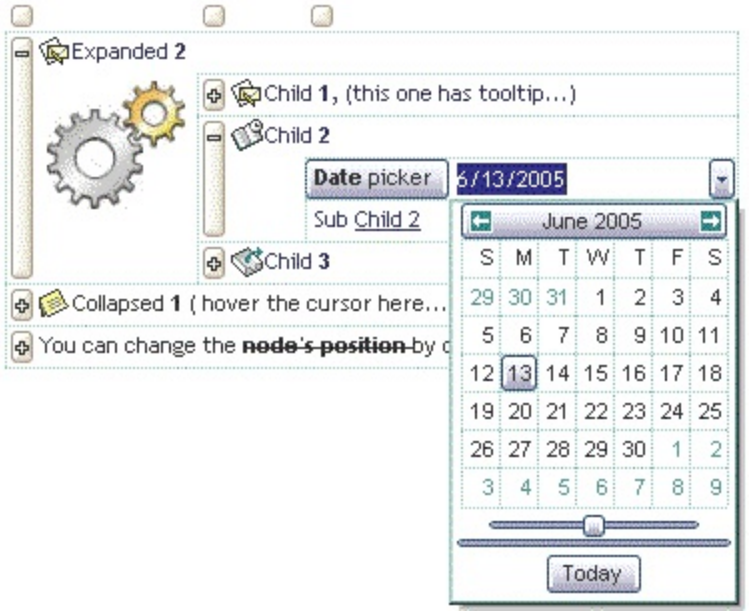
```
With XMLGrid1
   Dim i As Long
   For i = 0 To .VisibleNodeCount - 1
      Dim n As EXMLGRIDLibCtl.Node
      Set n = .NodeByPosition(i)
      Debug.Print n.Name
   Next
End With
```

property XMLGrid.VisualAppearance as Appearance

Retrieves the control's appearance.

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

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



property XMLGrid.VisualDesign as String

Invokes the control's VisualAppearance designer.

Type	Description
String	A String expression that encodes the control's Visual Appearance.

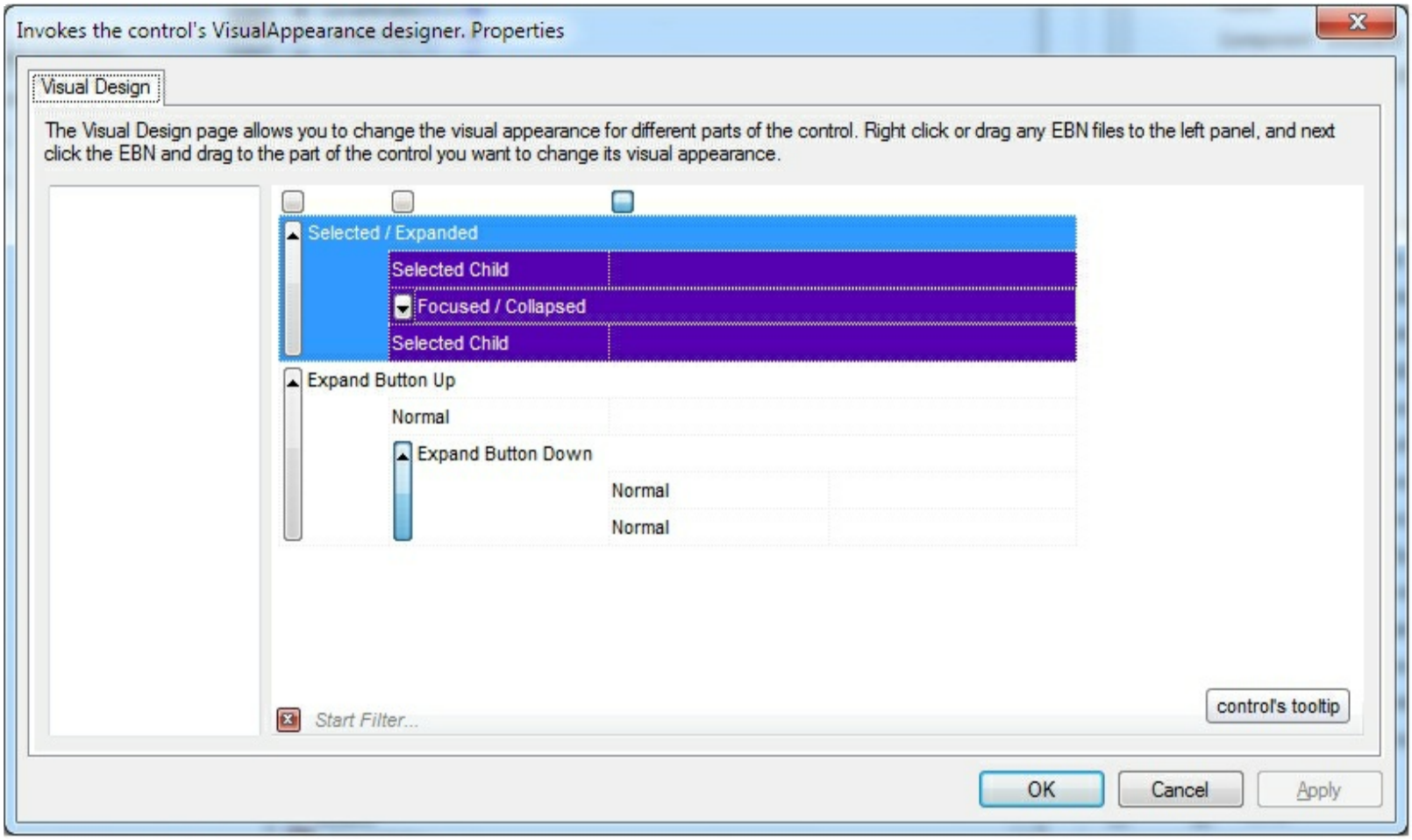
By default, the VisualDesign property is "". The VisualDesign property helps you to define fast and easy the control's visual appearance using the XP-Theme elements or EBN objects. The VisualDesign property can be accessed on design mode, and it can be used to design the visual appearance of different 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.

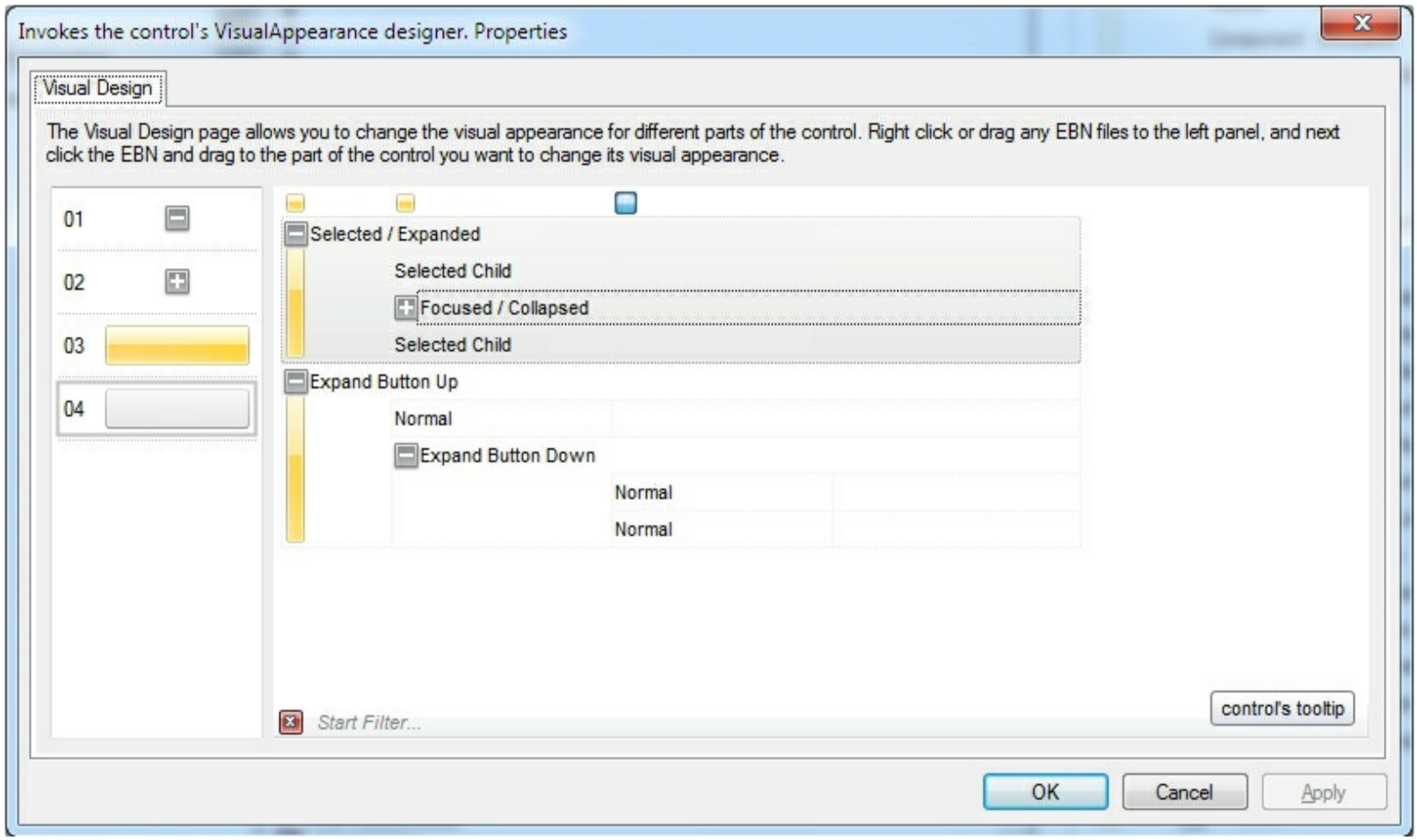
- Click here to watch a movie on how you define the control's visual appearance using the XP-Theme
- Click here to watch a movie on how you define the control's visual appearance using the EBN files.

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):



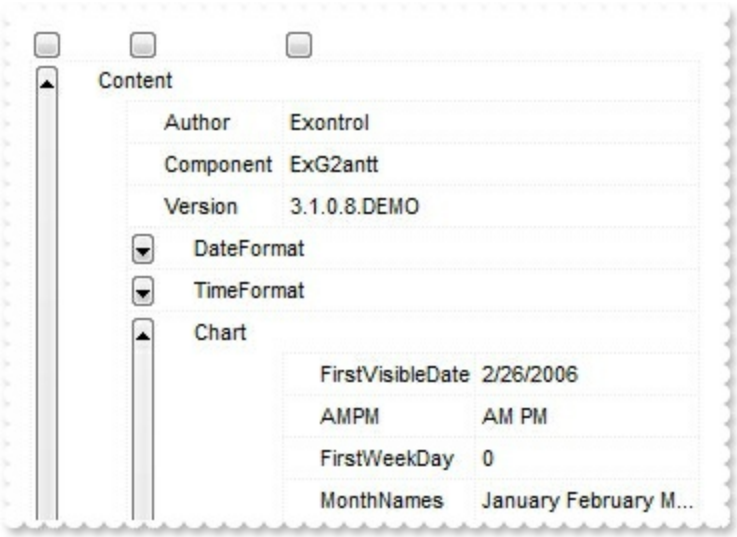
The following picture shows the control's VisualDesign form after applying some EBN objects:



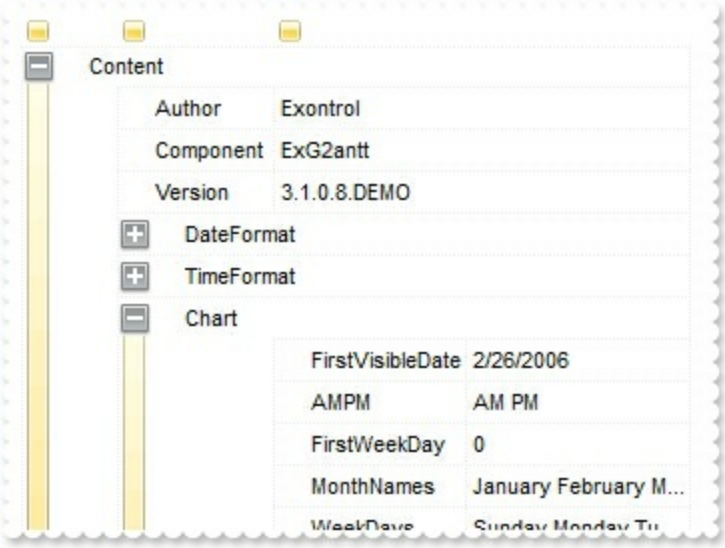
This layout generates the following code:

```
With XMLGrid1
   .VisualDesign =
"gBFLBWIgBAEHhEJAEGg7nBcHVJUAoABMIZ7/jEZf78jMJAkKAAEBkgAYOkACB8gAIxhEaGU
 & _
"JhDeDZZDYOwlgmQhghaGZmkmKhnhoZo5ioTYYk2WYEgEYAnGOGJlDkCQyECDq6ikNoCC
 & _
"okRpjBGSDoF4rASDoDuB0U4YQNBEEgOQQIGgiiSBoL0GYTAphUHaDhmTLwTBDGkAcMg6
 & _
"icBOPgdwcAPjjCSL4P4ixTDxH+BjHIaROBFAuAYKIgQMDHCiNEBYHhlBnHwPkEIjgJZVBGAsA
 & _
"MYP4LAzDyG4FEf4wxMjECyHwNGLx4j+DcOgboRwnjUHQMYcwjA2hiEMMYFAaxxBOgWA
 & _
"hATB9hRghg0AkC+BnBuAhAdh3gnAmh7BJArBgACBWACFmASBCAFAkAgBiBCACAllPg0gu
 & _
"gPAjhShWgzhsgKBPMnBygjAPBGhOhrAbAehdg1gsB9AIh5P1BjCcgxB1BDhnBWgvhvB7B3E

End With
```

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:


Content
Author
Exontrol
Component
ExG2antt
Version
3.1.0.8.DEMO
DateFormat
TimeFormat
Chart
FirstVisibleDate
2/26/2006
AMPM
AM PM
FirstWeekDay
0
MonthNames
January February M...
WeekDays
Sunday Monday Tu

EXMLGrid 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: {AC7F976E-48C3-4B0B-B952-45D92DFE7F3E}. The object's program identifier is: "Exontrol.XMLGrid". The /COM object module is: "EXMLGrid.dll"

The Exontrol's XMLGrid component supports the following events:

Name	Description
AddNode	Occurs when a node is added to the nodes collection.
AfterExpandNode	Notifies the application when a node is expanded or collapsed.
AnchorClick	Occurs when an anchor element is clicked.
BeforeExpandNode	Occurs when a node is about to be expanded or collapsed.
ButtonClick	Occurs when user clicks on the cell's button.
Change	Occurs when the user changes the cell's content.
Click	Occurs when the user presses and then releases the left mouse button over the control.
DblClick	Occurs when the user dblclk the left mouse button over an object.
Edit	Occurs just before editing the focused node.
EditClose	Occurs when the edit operation ends.
EditOpen	Occurs when the edit operation starts.
Event	Notifies the application once the control fires an event.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
MouseDown	Occurs when the user presses a mouse button.
MouseMove	Occurs when the user moves the mouse.
MouseUp	Occurs when the user releases a mouse button.
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
	Occurs when a source component is dropped onto a

OLEDragDrop	target component when the source component determines that a drop can occur.
OLEDragOver	Occurs when one component is dragged over another.
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.
RemoveNode	Occurs when a node is removed from the nodes collection.
ResizeLevel	Occurs when the user resizes the level.
ScrollButtonClick	Occurs when the user clicks a button in the scrollbar.
SelectionChanged	Fires when the user changes the selection.
UserEditorClose	Fired the user editor is about to be opened.
UserEditorOleEvent	Occurs when an user editor fires an event.
UserEditorOpen	Occurs when an user editor is about to be opened.

event AddNode (NewNode as Node)

Occurs when a node is added to the nodes collection.

Type	Description
NewNode as Node	A Node object being inserted.

The AddNode event notifies your application that user adds a new node. Use the Add method to insert a new node to the Nodes collection. Use the Nodes property to access the control's nodes collection. Use the AddNode event to associate extra data to the newly inserted node. Use the Add method to add new type of editors to the control. Use the Editor property to assign an editor to a node.

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

C#
```
private void AddNode(object sender,exontrol.EXMLGRIDLib.Node NewNode)
{
}
```

VB
```
Private Sub AddNode(ByVal sender As System.Object,ByVal NewNode As
exontrol.EXMLGRIDLib.Node) Handles AddNode
End Sub
```

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

C#
```
private void AddNode(object sender,
AxEXMLGRIDLib._IXMLGridEvents_AddNodeEvent e)
{
}
```

C++
```
void OnAddNode(LPDISPATCH NewNode)
{
}
```

C++ Builder
```
void __fastcall AddNode(TObject *Sender,Exmlgridlib_tlb::INode *NewNode)
{
}
```

Delphi
```
procedure AddNode(ASender: TObject; NewNode : INode);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AddNode(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_AddNodeEvent);
begin
end;
```

Powe…

```
begin event AddNode(oleobject NewNode)
end event AddNode
```

VB.NET

```
Private Sub AddNode(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_AddNodeEvent) Handles AddNode
End Sub
```

VB6

```
Private Sub AddNode(ByVal NewNode As EXMLGRIDLibCtl.INode)
End Sub
```

VBA

```
Private Sub AddNode(ByVal NewNode As Object)
End Sub
```

VFP

```
LPARAMETERS NewNode

```

Xbas…

```
PROCEDURE OnAddNode(oXMLGrid,NewNode)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComAddNode Variant llNewNode
    Forward Send OnComAddNode llNewNode
End_Procedure
```

Visual Objects

```
METHOD OCX_AddNode(NewNode) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddNode(COM _NewNode)
{
}
```

XBasic

```
function AddNode as v (NewNode as OLE::Exontrol.XMLGrid.1::INode)
end function
```

dBASE

```
function nativeObject_AddNode(NewNode)
return
```

The following VB sample assigns a default editors to all nodes, using the AddNode event:

```
Private Sub Form_Load()
   With XMLGrid1
      .BeginUpdate

      With .Editors
         With .Add("Edit", EditType)
            .Appearance = SingleApp
         End With
      End With

      With .Nodes
         With .Add("Root").Nodes
            .Add "Child 1", "text1"
            .Add "Child 2", "text2"
         End With
      End With
      .EndUpdate
   End With
End Sub

Private Sub XMLGrid1_AddNode(ByVal NewNode As EXMLGRIDLibCtl.INode)
   NewNode.Editor = "Edit"
```

```
End Sub
```

The following C++ sample assigns a default editors to all nodes, using the AddNode event:

```
#include "Node.h"
void OnAddNodeXmlgrid1(LPDISPATCH NewNode)
{
   CNode node( NewNode ); node.m_bAutoRelease = FALSE;
   node.SetEditor( COleVariant( "Edit" ) );
}
```

The following VB.NET sample assigns a default editors to all nodes, using the AddNode event:

```
Private Sub AxXMLGrid1_AddNode(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_AddNodeEvent) Handles AxXMLGrid1.AddNode
   e.newNode.Editor = "Edit"
End Sub
```

The following C# sample assigns a default editors to all nodes, using the AddNode event:

```
private void axXMLGrid1_AddNode(object sender,
AxEXMLGRIDLib._IXMLGridEvents_AddNodeEvent e)
{
   e.newNode.Editor = "Edit";
}
```

The following VFP sample assigns a default editors to all nodes, using the AddNode event:

```
*** ActiveX Control Event ***
LPARAMETERS newnode

with newnode
   .Editor = "Edit"
endwith
```

event AfterExpandNode (Node as Node)

Notifies the application when a node is expanded or collapsed.

Type	Description
Node as Node	A Node object being expanded or collapsed.

Use the AfterExpandNode event to notify your application that a node is expanded or collapsed. Use the Expanded property to programmatically expand or collapse a node. Use the Expanded property to find out if a node is expanded or collapsed. Use the ExpandAll method to expand all nodes in the control. Use the CollapseAll method to collapse all nodes in the control. Use the ExpandAll method to expand all child nodes of specified node. Use the CollapseAll method to collapse all child nodes of specified node. Use the BeforeExpandNode event to prevent expanding or collapsing a node.

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

C#
```
private void AfterExpandNode(object sender,exontrol.EXMLGRIDLib.Node Node)
{
}
```

VB
```
Private Sub AfterExpandNode(ByVal sender As System.Object,ByVal Node As exontrol.EXMLGRIDLib.Node) Handles AfterExpandNode
End Sub
```

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

C#
```
private void AfterExpandNode(object sender, AxEXMLGRIDLib._IXMLGridEvents_AfterExpandNodeEvent e)
{
}
```

C++
```
void OnAfterExpandNode(LPDISPATCH Node)
{
}
```

C++ Builder
```
void __fastcall AfterExpandNode(TObject *Sender,Exmlgridlib_tlb::INode *Node)
{
}
```

Delphi
```
procedure AfterExpandNode(ASender: TObject; Node : INode);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterExpandNode(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_AfterExpandNodeEvent);
begin
end;
```

Powe…

```
begin event AfterExpandNode(oleobject Node)
end event AfterExpandNode
```

VB.NET

```
Private Sub AfterExpandNode(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_AfterExpandNodeEvent) Handles
AfterExpandNode
End Sub
```

VB6

```
Private Sub AfterExpandNode(ByVal Node As EXMLGRIDLibCtl.INode)
End Sub
```

VBA

```
Private Sub AfterExpandNode(ByVal Node As Object)
End Sub
```

VFP

```
LPARAMETERS Node

```

Xbas…

```
PROCEDURE OnAfterExpandNode(oXMLGrid,Node)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComAfterExpandNode Variant llNode
    Forward Send OnComAfterExpandNode llNode
End_Procedure
```

Visual Objects

```
METHOD OCX_AfterExpandNode(Node) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AfterExpandNode(COM _Node)
{
}
```

XBasic

```
function AfterExpandNode as v (Node as OLE::Exontrol.XMLGrid.1::INode)
end function
```

dBASE

```
function nativeObject_AfterExpandNode(Node)
return
```

The following VB sample displays the caption of the node being expanded or collapsed:

```
Private Sub XMLGrid1_AfterExpandNode(ByVal Node As EXMLGRIDLibCtl.INode)
    Debug.Print "The '" & Node.Name & "' is " & IIf(Node.Expanded, "expanded",
"collapsed") & "."
End Sub
```

The following C++ sample displays the caption of the node being expanded or collapsed:

```
#include "Node.h"
void OnAfterExpandNodeXmlgrid1(LPDISPATCH Node)
{
    CNode node( Node ); node.m_bAutoRelease = FALSE;
    CString strFormat, strName = node.GetName();
    strFormat.Format( "The %s is %s.", strName, (node.GetExpanded() ? "expanded" :
"collapsed" ) );
    OutputDebugString( strFormat );
}
```

The following VB.NET sample displays the caption of the node being expanded or collapsed:

```
Private Sub AxXMLGrid1_AfterExpandNode(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_AfterExpandNodeEvent) Handles
AxXMLGrid1.AfterExpandNode
    Dim strMessage As String = "The " + e.node.Name + " is "
    strMessage = strMessage + (IIf(e.node.Expanded, "expanded", "collapsed"))
    Debug.Write(strMessage)
End Sub
```

The following C# sample displays the caption of the node being expanded or collapsed:

```
private void axXMLGrid1_AfterExpandNode(object sender,
AxEXMLGRIDLib._IXMLGridEvents_AfterExpandNodeEvent e)
{
    String strMessage = "The " + e.node.Name + " is ";
    strMessage += (e.node.Expanded ? "expanded" : "collapsed");
    System.Diagnostics.Debug.Write(strMessage);
}
```

The following VFP sample displays the caption of the node being expanded or collapsed:

```
*** ActiveX Control Event ***
LPARAMETERS node

with node
    s = "The " + .Name + " is "
    if ( .Expanded )
        s = s + "expanded"
    else
        s = s + "collapsed"
    endif
    wait window nowait s
endwith
```

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*, 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 *<a 1;yourextradata>anchor*, the AnchorID parameter of the AnchorClick event is 1, and the Options parameter is "yourextradata". Use the AnchorFromPoint property to retrieve the identifier of the anchor element from the cursor.

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

C#

```
private void AnchorClick(object sender,string AnchorID,string Options)
{
}
```

VB

```
Private Sub AnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles AnchorClick
End Sub
```

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

C#

```
private void AnchorClick(object sender,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_AnchorClickEvent);
begin
end;
```

Powe…

```
begin event AnchorClick(string AnchorID,string Options)
end event AnchorClick
```

VB.NET

```
Private Sub AnchorClick(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_AnchorClickEvent) Handles AnchorClick
End Sub
```

VB6

```
Private Sub AnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

VBA

```
Private Sub AnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

VFP

```
LPARAMETERS AnchorID,Options

```

Xbas…

```
PROCEDURE OnAnchorClick(oXMLGrid,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 llAnchorID String llOptions
    Forward Send OnComAnchorClick llAnchorID llOptions
End_Procedure
```

Visual Objects

```
METHOD OCX_AnchorClick(AnchorID,Options) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AnchorClick(str _AnchorID,str _Options)
{
}
```

XBasic

```
function AnchorClick as v (AnchorID as C,Options as C)
end function
```

dBASE

```
function nativeObject_AnchorClick(AnchorID,Options)
return
```

event BeforeExpandNode (Node as Node, Cancel as Variant)

Occurs when a node is about to be expanded or collapsed.

Type	Description
Node as Node	A Node object being expanded or collapsed.
Cancel as Variant	A boolean expression that indicates whether the operation is canceled or not.

Use the BeforeExpandNode event to notify your application that the user is about to expand or collapse a node. Use the Expanded property to expand or collapse a node. Use the HasChilds property to display expanding/collapsing buttons for a node to build your virtual tree. You can use the BeforeExpandNode event to cancel expanding specified nodes.

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

C#
```
private void BeforeExpandNode(object sender,exontrol.EXMLGRIDLib.Node Node,ref object Cancel)
{
}
```

VB
```
Private Sub BeforeExpandNode(ByVal sender As System.Object,ByVal Node As exontrol.EXMLGRIDLib.Node,ByRef Cancel As Object) Handles BeforeExpandNode
End Sub
```

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

C#
```
private void BeforeExpandNode(object sender, AxEXMLGRIDLib._IXMLGridEvents_BeforeExpandNodeEvent e)
{
}
```

C++
```
void OnBeforeExpandNode(LPDISPATCH Node,VARIANT FAR* Cancel)
{
}
```

C++ Builder
```
void __fastcall BeforeExpandNode(TObject *Sender,Exmlgridlib_tlb::INode *Node,Variant * Cancel)
{
}
```

Delphi

```
procedure BeforeExpandNode(ASender: TObject; Node : INode;var Cancel :
OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeExpandNode(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_BeforeExpandNodeEvent);
begin
end;
```

Powe…

```
begin event BeforeExpandNode(oleobject Node,any Cancel)
end event BeforeExpandNode
```

VB.NET

```
Private Sub BeforeExpandNode(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_BeforeExpandNodeEvent) Handles
BeforeExpandNode
End Sub
```

VB6

```
Private Sub BeforeExpandNode(ByVal Node As EXMLGRIDLibCtl.INode,Cancel As
Variant)
End Sub
```

VBA

```
Private Sub BeforeExpandNode(ByVal Node As Object,Cancel As Variant)
End Sub
```

VFP

```
LPARAMETERS Node,Cancel

```

Xbas…

```
PROCEDURE OnBeforeExpandNode(oXMLGrid,Node,Cancel)
RETURN
```

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

Java…

```
<SCRIPT EVENT="BeforeExpandNode(Node,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function BeforeExpandNode(Node,Cancel)
```

```
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComBeforeExpandNode Variant llNode Variant llCancel
    Forward Send OnComBeforeExpandNode llNode llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_BeforeExpandNode(Node,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_BeforeExpandNode(COM _Node,COMVariant /*variant*/ _Cancel)
{
}
```

XBasic

```
function BeforeExpandNode as v (Node as OLE::Exontrol.XMLGrid.1::INode,Cancel
as A)
end function
```

dBASE

```
function nativeObject_BeforeExpandNode(Node,Cancel)
return
```

The following VB sample adds new child nodes to the node that's about to be expanded:

```
Private Sub XMLGrid1_BeforeExpandNode(ByVal Node As EXMLGRIDLibCtl.INode, Cancel
As Variant)
   If Not Node.Expanded Then
      With Node.Nodes
         With .Add("New Node")
            .HasChilds = True
         End With
      End With
   End If
End Sub
```

The following C++ sample adds new child nodes to the node that's about to be expanded:

```
#include "Node.h"
#include "Nodes.h"
```

```
void OnBeforeExpandNodeXmlgrid1(LPDISPATCH Node, VARIANT FAR* Cancel)
{
   if ( IsWindow( m_xmlgrid.m_hWnd) )
   {
      CNode node( Node ); node.m_bAutoRelease = FALSE;
      if ( !node.GetExpanded() )
      {
         COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
         CNode newNode = node.GetNodes().Add( "New Node", vtMissing, vtMissing );
         newNode.SetHasChilds( TRUE );
      }
   }
}
```

The following VB.NET sample adds new child nodes to the node that's about to be expanded:

```
Private Sub AxXMLGrid1_BeforeExpandNode(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_BeforeExpandNodeEvent) Handles
AxXMLGrid1.BeforeExpandNode
   If Not e.node.Expanded Then
      With e.node.Nodes
         With .Add("New Node")
            .HasChilds = True
         End With
      End With
   End If
End Sub
```

The following C# sample adds new child nodes to the node that's about to be expanded:

```
private void axXMLGrid1_BeforeExpandNode(object sender,
AxEXMLGRIDLib._IXMLGridEvents_BeforeExpandNodeEvent e)
{
   if (!e.node.Expanded)
   {
      EXMLGRIDLib.Nodes nodes = e.node.Nodes;
      nodes.Add( "New Node", null, null ).HasChilds = true;
```

```
    }
}
```

The following VFP sample adds new child nodes to the node that's about to be expanded:

```
*** ActiveX Control Event ***
LPARAMETERS node, cancel

with node
    If !.Expanded Then
        With .Nodes
            With .Add("New Node")
                .HasChilds = .t.
            EndWith
        EndWith
    EndIf
endwith
```

event ButtonClick (Node as Node, Key as Variant)

Occurs when user clicks on the cell's button.

Type	Description
Node as Node	A Node object being clicked.
Key as Variant	A Variant expression that indicates the key of the button inside the node being clicked.

Use the ButtonClick event to notify your application that the user clicks a button inside a node. Use the AddButton method to add new buttons to an editor. Use the Editors property to access the control's Editors collection. Use the Add method to add new type of editors to the control. Use the Editor property to assign an editor to a node.

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

C#

```
private void ButtonClick(object sender,exontrol.EXMLGRIDLib.Node Node,object Key)
{
}
```

VB

```
Private Sub ButtonClick(ByVal sender As System.Object,ByVal Node As exontrol.EXMLGRIDLib.Node,ByVal Key As Object) Handles ButtonClick
End Sub
```

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

C#

```
private void ButtonClick(object sender, AxEXMLGRIDLib._IXMLGridEvents_ButtonClickEvent e)
{
}
```

C++

```
void OnButtonClick(LPDISPATCH Node,VARIANT Key)
{
}
```

C++ Builder

```
void __fastcall ButtonClick(TObject *Sender,Exmlgridlib_tlb::INode *Node,Variant Key)
{
}
```

Delphi
```
procedure ButtonClick(ASender: TObject; Node : INode;Key : OleVariant);
begin
end;
```

Delphi 8 (.NET only)
```
procedure ButtonClick(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_ButtonClickEvent);
begin
end;
```

Powe…
```
begin event ButtonClick(oleobject Node,any Key)
end event ButtonClick
```

VB.NET
```
Private Sub ButtonClick(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_ButtonClickEvent) Handles ButtonClick
End Sub
```

VB6
```
Private Sub ButtonClick(ByVal Node As EXMLGRIDLibCtl.INode,ByVal Key As
Variant)
End Sub
```

VBA
```
Private Sub ButtonClick(ByVal Node As Object,ByVal Key As Variant)
End Sub
```

VFP
```
LPARAMETERS Node,Key

```

Xbas…
```
PROCEDURE OnButtonClick(oXMLGrid,Node,Key)
RETURN
```

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

Java…
```
<SCRIPT EVENT="ButtonClick(Node,Key)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function ButtonClick(Node,Key)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComButtonClick Variant llNode Variant llKey
    Forward Send OnComButtonClick llNode llKey
End_Procedure
```

Visual Objects

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

X++

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

XBasic

```
function ButtonClick as v (Node as OLE::Exontrol.XMLGrid.1::INode,Key as A)
end function
```

dBASE

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

The following VB sample displays a message box when user clicks the 'A' button:

```
Private Sub Form_Load()
   With XMLGrid1
      .BeginUpdate
         With .Editors
            With .Add("Spin", SpinType)
               .ButtonWidth = 18
               .AddButton "A", 1
            End With
         End With
         With .Nodes
            With .Add("Spin", 1)
               .Editor = "Spin"
            End With
         End With
      .EndUpdate
   End With
End Sub
```

```
Private Sub XMLGrid1_ButtonClick(ByVal Node As EXMLGRIDLibCtl.INode, ByVal Key As
Variant)
   If Key = "A" Then
      MsgBox "You have clicked the 'A' button."
   End If
End Sub
```

The following C++ sample displays a message box when user clicks the 'A' button:

```
#include "Node.h"
void OnButtonClickXmlgrid1(LPDISPATCH Node, const VARIANT FAR& Key)
{
   CNode node( Node ); node.m_bAutoRelease = FALSE;
   if ( V2S( &Key ) == "A" )
      MessageBox( "Click the button" ) ;
}
```

where the VS2 function converts a VARIANT expression to a string expression:

```
static CString V2S( const VARIANT* pvtValue )
{
   COleVariant vtString;
   vtString.ChangeType( VT_BSTR, (VARIANT*)pvtValue );
   return V_BSTR( &vtString );
}
```

The following VB.NET sample displays a message box when user clicks the 'A' button:

```
Private Sub AxXMLGrid1_ButtonClick(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_ButtonClickEvent) Handles AxXMLGrid1.ButtonClick
   If e.key = "A" Then
      MsgBox("You have clicked the 'A' button.")
   End If
End Sub
```

The following C# sample displays a message box when user clicks the 'A' button:

```
private void axXMLGrid1_ButtonClick(object sender,
AxEXMLGRIDLib._IXMLGridEvents_ButtonClickEvent e)
```

```
{
    if (e.key.ToString() == "A")
        MessageBox.Show("The user clicks the 'A' button. ");
}
```

The following VFP sample displays a message box when user clicks the 'A' button:

```
*** ActiveX Control Event ***
LPARAMETERS node, key

if ( key = "A" )
    wait window nowait "The uuser clicks the 'A' button. "
endif
```

event Change (Node as Node, NewValue as Variant)

Occurs when the user changes the cell's content.

Type	Description
Node as Node	A Node object whose value is chaning.
NewValue as Variant	A Variant expression that indicates the newly node's value.

Use the Change event to notify your application when the node's value is changed. The Change event notifies that the editing focused node ended. Use the Value property to assign a value to a node. Use the Name property to assign a caption to a node.

The NewValue parameter indicates the newly node's value before assigning it to the Value property. You can use the Value property to get the old node's value.

The edit events are fired in the following order:

1. Edit event. Prevents editing nodes, before showing the node's editor.
2. EditOpen event. The edit operation started, the node's editor is shown. The Editing property gives the window's handle of the built-in editor being shown.
3. Change event. The Change event is fired only if the user types ENTER key, the user selects a new value from a predefined data list, or focus a new node.
4. EditClose event. The node's editor is hidden and closed.

If a node has an editor assigned the node's editor is applied to the:

- Name property, if the node contains child node.
- Value property, if the node contains no child node.

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

C#

```
private void Change(object sender,exontrol.EXMLGRIDLib.Node Node,ref object NewValue)
{
}
```

VB

```
Private Sub Change(ByVal sender As System.Object,ByVal Node As exontrol.EXMLGRIDLib.Node,ByRef NewValue As Object) Handles Change
End Sub
```

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

C#

```
private void Change(object sender,
AxEXMLGRIDLib._IXMLGridEvents_ChangeEvent e)
{
}
```

C++

```
void OnChange(LPDISPATCH Node,VARIANT FAR* NewValue)
{
}
```

C++ Builder

```
void __fastcall Change(TObject *Sender,Exmlgridlib_tlb::INode *Node,Variant *
NewValue)
{
}
```

Delphi

```
procedure Change(ASender: TObject; Node : INode;var NewValue : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

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

Powe…

```
begin event Change(oleobject Node,any NewValue)
end event Change
```

VB.NET

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

VB6

```
Private Sub Change(ByVal Node As EXMLGRIDLibCtl.INode,NewValue As Variant)
End Sub
```

VBA

```
Private Sub Change(ByVal Node As Object,NewValue As Variant)
End Sub
```

VFP

```
LPARAMETERS Node,NewValue
```

```
```

Xbas…
```
PROCEDURE OnChange(oXMLGrid,Node,NewValue)
RETURN
```

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

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

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

Visual Data…
```
Procedure OnComChange Variant llNode Variant llNewValue
    Forward Send OnComChange llNode llNewValue
End_Procedure
```

Visual Objects
```
METHOD OCX_Change(Node,NewValue) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_Change(COM _Node,COMVariant /*variant*/ _NewValue)
{
}
```

XBasic
```
function Change as v (Node as OLE::Exontrol.XMLGrid.1::INode,NewValue as A)
end function
```

dBASE
```
function nativeObject_Change(Node,NewValue)
return
```

The following VB sample assign a drop down editor to a cell and displays the newly value when user selects a new value from the drop down portion of the editor:

```
Private Sub Form_Load()
   With XMLGrid1
      .BeginUpdate
```

```
        With .Editors
            With .Add("DD", DropDownListType)
                .AddButton "A", 1
                .AddButton "B", 1, RightAlignment
                .AddItem 1, "<b>1</b> One"
                .AddItem 2, "<b>2</b> One"
                .AddItem 3, "<b>3</b> One"
            End With
        End With
        With .Nodes
            With .Add("Select", 1)
                .Editor = "DD"
            End With
        End With
    .EndUpdate
  End With
End Sub

Private Sub XMLGrid1_Change(ByVal Node As EXMLGRIDLibCtl.INode, NewValue As
Variant)
  Debug.Print "NewValue = " & NewValue
End Sub
```

The following C++ sample displays the value that user changes:

```
#include "Node.h"
void OnChangeXmlgrid1(LPDISPATCH Node, VARIANT FAR* NewValue)
{
  CNode node( Node ); node.m_bAutoRelease = FALSE;
  CString strNewValue = V2S( NewValue );
  OutputDebugString( strNewValue );
}
```

where the V2S function converts a VARIANT expression to a string expression:

```
static CString V2S( const VARIANT* pvtValue )
{
  COleVariant vtString;
```

```
    vtString.ChangeType( VT_BSTR, (VARIANT*)pvtValue );
    return V_BSTR( &vtString );
}
```

The following VB.NET sample displays the value that user changes:

```
Private Sub AxXMLGrid1_Change(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_ChangeEvent) Handles AxXMLGrid1.Change
    Debug.Write(e.newValue.ToString())
End Sub
```

The following C# sample displays the value that user changes:

```
private void axXMLGrid1_Change(object sender,
AxEXMLGRIDLib._IXMLGridEvents_ChangeEvent e)
{
    System.Diagnostics.Debug.Write(e.newValue.ToString());
}
```

The following VFP sample displays the value that user chages:

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

wait window nowait newValue
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the control.

Type	Description

The Click event is fired when the user releases the left mouse button over the control. Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

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

C#

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

VB

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

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

C#

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

C++

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

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

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

VB.NET

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

VB6

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

VBA

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

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnClick(oXMLGrid)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComClick
    Forward Send OnComClick
```

```
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_Click()
{
}
```

XBasic

```
function Click as v ()
end function
```

dBASE

```
function nativeObject_Click()
return
```

event 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 dbl clicks on the control. Use the DblClick event to notify your application that a cell has been double-clicked. Use the NodeFromPoint method to get the node from cursor. Use the ExpandOnDblClk property to disable expanding or collapsing a node when user double clicks a node.

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,
AxEXMLGRIDLib._IXMLGridEvents_DblClickEvent e)
{
}
```

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

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

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

Delphi 8 (.NET only)
```
procedure DblClick(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_DblClickEvent);
begin
end;
```

Powe…
```
begin event DblClick(integer Shift,long X,long Y)
end event DblClick
```

VB.NET
```
Private Sub DblClick(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_DblClickEvent) Handles DblClick
End Sub
```

VB6
```
Private Sub DblClick(Shift As Integer,X As Single,Y As Single)
End Sub
```

VBA
```
Private Sub DblClick(ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub
```

VFP
```
LPARAMETERS Shift,X,Y

```

Xbas…
```
PROCEDURE OnDblClick(oXMLGrid,Shift,X,Y)
RETURN
```

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

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

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

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

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

The following VB sample displays the node being double clicked:

```
Private Sub XMLGrid1_DblClick(Shift As Integer, X As Single, Y As Single)
  With XMLGrid1
    Dim n As EXMLGRIDLibCtl.Node
    Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not n Is Nothing Then
      Debug.Print "You have clicked the '" & n.Name & "'."
    End If
  End With
End Sub
```

The following C++ sample displays the node being double clicked:

```
#include "Node.h"
void OnDblClickXmlgrid1(short Shift, long X, long Y)
{
   CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
   if ( node.m_lpDispatch != NULL )
   {
      CString strName = node.GetName();
      OutputDebugString( strName );
   }
}
```

The following VB.NET sample displays the node being double clicked:

```
Private Sub AxXMLGrid1_DblClick(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_DblClickEvent) Handles AxXMLGrid1.DblClick
   With AxXMLGrid1
      Dim n As EXMLGRIDLib.Node
      n = .get_NodeFromPoint(e.x, e.y)
      If Not n Is Nothing Then
         Debug.Print("You have clicked the '" & n.Name & "'.")
      End If
   End With
End Sub
```

The following C# sample displays the node being double clicked:

```
private void axXMLGrid1_DblClick(object sender,
AxEXMLGRIDLib._IXMLGridEvents_DblClickEvent e)
{
   EXMLGRIDLib.Node node = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
   if (node != null)
      System.Diagnostics.Debug.Write(node.Name);
}
```

The following VFP sample displays the node being double clicked:

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

```
with thisform.XMLGrid1
    n = .NodeFromPoint(x, y )
    if ( !isnull(n) )
        wait window nowait n.Name
    endif
endwith
```

event Edit (Node as Node, Cancel as Boolean)

Occurs just before editing the focused node.

Type	Description
Node as Node	A Node object being edited.
Cancel as Boolean	A boolean expression that indicates whether the edit operation is canceled.

The Edit event is fired when the editing operation is about to begin. Use the Edit event to disable editing specific nodes. Use the Editor property to assign an editor to a node. Use the Editors property to access the control's Editors collection. Use the Edit method to programmatically edit a node, if the AutoEdit property is False.

If a node has an editor assigned the node's editor is applied to the:

- Name property, if the node contains child node.
- Value property, if the node contains no child node.

The edit events are fired in the following order:

1. Edit event. Prevents editing nodes, before showing the node's editor.
2. EditOpen event. The edit operation started, the node's editor is shown. The Editing property gives the window's handle of the built-in editor being shown.
3. Change event. The Change event is fired only if the user types ENTER key, the user selects a new value from a predefined data list, or focus a new node.
4. EditClose event. The node's editor is hidden and closed.

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

C#
```
private void EditEvent(object sender,exontrol.EXMLGRIDLib.Node Node,ref bool Cancel)
{
}
```

VB
```
Private Sub EditEvent(ByVal sender As System.Object,ByVal Node As exontrol.EXMLGRIDLib.Node,ByRef Cancel As Boolean) Handles EditEvent
End Sub
```

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

C#

```
private void EditEvent(object sender, AxEXMLGRIDLib._IXMLGridEvents_EditEvent
e)
{
}
```

C++

```
void OnEdit(LPDISPATCH Node,BOOL FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall Edit(TObject *Sender,Exmlgridlib_tlb::INode *Node,VARIANT_BOOL
* Cancel)
{
}
```

Delphi

```
procedure Edit(ASender: TObject; Node : INode;var Cancel : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure EditEvent(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_EditEvent);
begin
end;
```

Powe…

```
begin event Edit(oleobject Node,boolean Cancel)
end event Edit
```

VB.NET

```
Private Sub EditEvent(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_EditEvent) Handles EditEvent
End Sub
```

VB6

```
Private Sub Edit(ByVal Node As EXMLGRIDLibCtl.INode,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub Edit(ByVal Node As Object,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS Node,Cancel
```

Xbas…
```
PROCEDURE OnEdit(oXMLGrid,Node,Cancel)
RETURN
```

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

Java…
```
<SCRIPT EVENT="Edit(Node,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function Edit(Node,Cancel)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComEdit Variant llNode Boolean llCancel
    Forward Send OnComEdit llNode llCancel
End_Procedure
```

Visual Objects
```
METHOD OCX_Edit(Node,Cancel) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_Edit(COM _Node,COMVariant /*bool*/ _Cancel)
{
}
```

XBasic
```
function Edit as v (Node as OLE::Exontrol.XMLGrid.1::INode,Cancel as L)
end function
```

dBASE
```
function nativeObject_Edit(Node,Cancel)
return
```

The following VB sample disables editing nodes that contain child nodes (parent nodes):

```
Private Sub Form_Load()
   With XMLGrid1
      .BeginUpdate
         .AutoEdit = True
         .Editors.Add "Edit", EditType
```

```
        With .Nodes
            With .Add("Root").Nodes
                .Add "Child 1", "text1"
                .Add "Child 2", "text2"
            End With
        End With
    .EndUpdate
  End With
End Sub

Private Sub XMLGrid1_AddNode(ByVal NewNode As EXMLGRIDLibCtl.INode)
  NewNode.Editor = "Edit"
End Sub

Private Sub XMLGrid1_Edit(ByVal Node As EXMLGRIDLibCtl.INode, Cancel As Boolean)
  Cancel = Not Node.Nodes.Count = 0
End Sub
```

The following C++ sample disables editing nodes that contain child nodes (parent nodes):

```
#include "Node.h"
void OnEditXmlgrid1(LPDISPATCH Node, BOOL FAR* Cancel)
{
  CNode node( Node ); node.m_bAutoRelease = FALSE;
  if ( node.GetNodes().GetCount() != 0 )
    *Cancel = TRUE;
}
```

The following VB.NET sample disables editing nodes that contain child nodes (parent nodes):

```
Private Sub AxXMLGrid1_EditEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_EditEvent) Handles AxXMLGrid1.EditEvent
  If (e.node.Nodes.Count <> 0) Then
    e.cancel = True
  End If
End Sub
```

The following C# sample disables editing nodes that contain child nodes (parent nodes):

```
private void axXMLGrid1_EditEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_EditEvent e)
{
    if (e.node.Nodes.Count != 0)
        e.cancel = true;
}
```

The following VFP sample disables editing nodes that contain child nodes (parent nodes):

```
*** ActiveX Control Event ***
LPARAMETERS node, cancel

if !( node.Nodes.Count = 0 )
    cancel = .t.
endif
```

event EditClose ()

Occurs when the edit operation ends.

Type	Description

The EditClose event notifies your application that the node's editor is hidden and closed. Use the FocusNode property to specify the control's focused node. Use the Editor property to assign an editor to a node. The Editing specifies the window's handle of the built-in editor while the control is running in edit mode. Use the AutoEdit property to specify whether the control starts editing the focused node.

The edit events are fired in the following order:

1. Edit event. Prevents editing nodes, before showing the node's editor.
2. EditOpen event. The edit operation started, the node's editor is shown. The Editing property gives the window's handle of the built-in editor being shown.
3. Change event. The Change event is fired only if the user types ENTER key, the user selects a new value from a predefined data list, or focus a new node.
4. EditClose event. The node's editor is hidden and closed.

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

C#
```
private void EditCloseEvent(object sender)
{
}
```

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

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

C#
```
private void EditCloseEvent(object sender, EventArgs e)
{
}
```

C++
```
void OnEditClose()
{
}
```

C++ Builder
```
void __fastcall EditClose(TObject *Sender)
{
}
```

Delphi
```
procedure EditClose(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)
```
procedure EditCloseEvent(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…
```
begin event EditClose()
end event EditClose
```

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

VB6
```
Private Sub EditClose()
End Sub
```

VBA
```
Private Sub EditClose()
End Sub
```

VFP
```
LPARAMETERS nop
```

Xbas…
```
PROCEDURE OnEditClose(oXMLGrid)
RETURN
```

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

Java…
```
<SCRIPT EVENT="EditClose()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function EditClose()
```

```
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEditClose
    Forward Send OnComEditClose
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_EditClose()
{
}
```

XBasic

```
function EditClose as v ()
end function
```

dBASE

```
function nativeObject_EditClose()
return
```

The following VB sample displays a message when an editor is closed:

```
Private Sub XMLGrid1_EditClose()
    Debug.Print "XMLGrid1_EditClose event is fired"
End Sub
```

The following C++ sample displays a message when an editor is closed:

```
void OnEditCloseXmlgrid1()
{
    OutputDebugString( "OnEditCloseXmlgrid1 is called." );
}
```

The following VB.NET sample displays a message when an editor is closed:

```
Private Sub AxXMLGrid1_EditClose(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxXMLGrid1.EditClose
    Debug.Print("AxXMLGrid1_EditClose is called.")
```

```
End Sub
```

The following C# sample displays a message when an editor is closed:

```
private void axXMLGrid1_EditClose(object sender, EventArgs e)
{
    System.Diagnostics.Debug.Write("axXMLGrid1_EditClose is called.");
}
```

The following VFP sample displays a message when an editor is closed:

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

wait window nowait "EditClose event is called."
```

event EditOpen ()

Occurs when the edit operation starts.

Type	Description

The EditOpen event notifies your application that the user starts editing a node. Use the FocusNode property to get the node being edited. Use the Editor property to assign an editor to a node. The Editing specifies the window's handle of the built-in editor while the control is running in edit mode. Use the AutoEdit property to specify whether the control starts editing the focused node.

The edit events are fired in the following order:

1. Edit event. Prevents editing nodes, before showing the node's editor.
2. EditOpen event. The edit operation started, the node's editor is shown. The Editing property gives the window's handle of the built-in editor being shown.
3. Change event. The Change event is fired only if the user types ENTER key, the user selects a new value from a predefined data list, or focus a new node.
4. EditClose event. The node's editor is hidden and closed.

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

C#

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

VB

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

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

C#

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

C++

```
void OnEditOpen()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event EditOpen()
end event EditOpen
```

VB.NET

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

VB6

```
Private Sub EditOpen()
End Sub
```

VBA

```
Private Sub EditOpen()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnEditOpen(oXMLGrid)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComEditOpen
    Forward Send OnComEditOpen
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_EditOpen()
{
}
```

XBasic

```
function EditOpen as v ()
end function
```

dBASE

```
function nativeObject_EditOpen()
return
```

The following VB sample displays a message when an editor is opened:

```
Private Sub XMLGrid1_EditOpen()
   Debug.Print "XMLGrid1_EditOpen event is fired"
End Sub
```

The following C++ sample displays a message when an editor is opened:

```
void OnEditOpenXmlgrid1()
{
   OutputDebugString( "OnEditOpenXmlgrid1 is called." );
}
```

The following VB.NET sample displays a message when an editor is opened:

```
Private Sub AxXMLGrid1_EditOpen(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxXMLGrid1.EditOpen
    Debug.Print("AxXMLGrid1_EditOpen is called.")
End Sub
```

The following C# sample displays a message when an editor is opened:

```
private void axXMLGrid1_EditOpen(object sender, EventArgs e)
{
    System.Diagnostics.Debug.Write("axXMLGrid1_EditOpen is called.");
}
```

The following VFP sample displays a message when an editor is opened:

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

wait window nowait "EditOpen event is called."
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

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

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

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

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

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

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

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   print exxmlgrid1.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
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

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

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, AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_EventEvent);
begin
end;
```

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

VB.NET
```
Private Sub Event(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,EventID)
RETURN
```

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

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

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

Visual Data…
```
Procedure OnComEvent Integer llEventID
   Forward Send OnComEvent llEventID
End_Procedure
```

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

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

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

```
end function
```

dBASE

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

event KeyDown (KeyCode as Integer, Shift as Integer)

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

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

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

Use the Editor property to assign an editor to a node. Use the Mask property to mask input characters while user types inside the node's editor. Use the Numeric property to specify whether the editor enables numeric values only. Use the Editing property to check whether the control is running in edit mode.

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,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,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
```

The following VB sample starts editing a node as soon as user presses the F2 key:

```
Private Sub XMLGrid1_KeyDown(KeyCode As Integer, Shift As Integer)
    With XMLGrid1
        If .Editing = 0 Then
```

```
        If KeyCode = vbKeyF2 Then
            .Edit
        End If
    End If
  End With
End Sub
```

The following C++ sample starts editing a node as soon as user presses the F2 key:

```
void OnKeyDownXmlgrid1(short FAR* KeyCode, short Shift)
{
  COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
  if ( m_xmlgrid.GetEditing() == 0 )
    if ( *KeyCode == VK_F2 )
      m_xmlgrid.Edit( vtMissing );
}
```

The following VB.NET sample starts editing a node as soon as user presses the F2 key:

```
Private Sub AxXMLGrid1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_KeyDownEvent) Handles AxXMLGrid1.KeyDownEvent
  If (AxXMLGrid1.Editing = 0) Then
    If (e.keyCode = Keys.F2) Then
      AxXMLGrid1.Edit()
    End If
  End If
End Sub
```

The following C# sample starts editing a node as soon as user presses the F2 key:

```
private void axXMLGrid1_KeyDownEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_KeyDownEvent e)
{
  if (axXMLGrid1.Editing == 0)
    if (e.keyCode == Convert.ToUInt16(Keys.F2))
      axXMLGrid1.Edit();
}
```

The following VFP sample starts editing a node as soon as user presses the F2 key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

with thisform.XMLGrid1
    if ( .Editing = 0 )
        if ( keycode = 113 )
            .Edit
        endif
    endif
endwith
```

event KeyPress (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	An integer that returns a standard numeric ANSI keycode

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

Use the Editor property to assign an editor to a node. Use the Mask property to mask input characters while user types inside the node's editor. Use the Numeric property to specify whether the editor enables numeric values only. Use the Editing property to check whether the control is running in edit mode.

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

C#

```
private void KeyPress(object sender,ref short KeyAscii)
{
}
```

VB

```
Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

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

C#

```
private void KeyPressEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_KeyPressEvent e)
{
}
```

C++

```
void OnKeyPress(short FAR* KeyAscii)
{
}
```

C++ Builder
```
void __fastcall KeyPress(TObject *Sender,short * KeyAscii)
{
}
```

Delphi
```
procedure KeyPress(ASender: TObject; var KeyAscii : Smallint);
begin
end;
```

Delphi 8 (.NET only)
```
procedure KeyPressEvent(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,KeyAscii)
RETURN
```

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

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

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

```
Function KeyPress(KeyAscii)
End Function
</SCRIPT>
```

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

event KeyUp (KeyCode as Integer, Shift as Integer)

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

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

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

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

C#

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

VB

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

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

C#

```
private void KeyUpEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_KeyUpEvent e)
{
}
```

C++

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

C++ Builder

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

```
}
```

Delphi

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

Delphi 8 (.NET only)

```
procedure KeyUpEvent(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,KeyCode,Shift)
RETURN
```

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

Java…

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

VBSc…

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

```
</SCRIPT>
```

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

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

Occurs when the user presses a mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the NodeFromPoint method to get the node from cursor.

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

C#

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

VB

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

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

C#

```
private void MouseDownEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,Button,Shift,X,Y)
RETURN
```

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

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

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

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

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

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

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

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

The following VB sample displays the node being clicked:

```
Private Sub XMLGrid1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    With XMLGrid1
        Dim n As EXMLGRIDLibCtl.Node
        Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
```

```
    If Not n Is Nothing Then
      Debug.Print "You have clicked the '" & n.Name & "'."
    End If
  End With
End Sub
```

The following C++ sample displays the node being clicked:

```
#include "Node.h"
void OnMouseDownXmlgrid1(short Button, short Shift, long X, long Y)
{
  CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
  if ( node.m_lpDispatch != NULL )
  {
    CString strName = node.GetName();
    OutputDebugString( strName );
  }
}
```

The following VB.NET sample displays the node being clicked:

```
Private Sub AxXMLGrid1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_MouseDownEvent) Handles
AxXMLGrid1.MouseDownEvent
  With AxXMLGrid1
    Dim n As EXMLGRIDLib.Node = .get_NodeFromPoint(e.x, e.y)
    If Not n Is Nothing Then
      Debug.Print("You have clicked the '" & n.Name & "'.")
    End If
  End With
End Sub
```

The following C# sample displays the node being clicked:

```
private void axXMLGrid1_MouseDownEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_MouseDownEvent e)
{
  EXMLGRIDLib.Node node = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
  if (node != null)
```

```
        System.Diagnostics.Debug.Write(node.Name);
}
```

The following VFP sample displays the node being clicked:

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

with thisform.XMLGrid1
    n = .NodeFromPoint(x, y )
    if ( !isnull(n) )
        wait window nowait n.Name
    endif
endwith
```

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

Occurs when the user moves the mouse.

Type	Description
Button as Integer	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders. Call the HitTest method to determine the location of the specified point relative to the client area of a xml grid view control. Use the NodeFromPoint property to get the node from the cursor.

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

C#

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

VB

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

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

C#

```
private void MouseMoveEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,Button,Shift,X,Y)
RETURN
```

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

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

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

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

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

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

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

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

The following VB sample prints the name of the node over the cursor:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    With XMLGrid1
        Dim n As EXMLGRIDLibCtl.Node
        Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
```

```
    If Not n Is Nothing Then
      Debug.Print "Hovers '" & n.Name & "'."
    End If
  End With
End Sub
```

The following VB sample displays the hit test code while user moves the mouse:

```
Private Sub XMLGrid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  With XMLGrid1
    Dim n As EXMLGRIDLibCtl.Node, h As EXMLGRIDLibCtl.HitTestEnum
    h = .HitTest(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, n)
    If Not h = 0 Then
      If (Not n Is Nothing) Then
        Debug.Print "Node = " & n.Name & " H = " & Hex(h)
      Else
        Debug.Print "H = " & Hex(h)
      End If
    End If
  End With
End Sub
```

The following C++ sample prints the name of the node from the cursor:

```
#include "Node.h"
void OnMouseMoveXmlgrid1(short Button, short Shift, long X, long Y)
{
  CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
  if ( node.m_lpDispatch != NULL )
  {
    CString strName = node.GetName();
    OutputDebugString( strName );
  }
}
```

The following VB.NET sample prints the name of the node from the cursor:

```
Private Sub AxXMLGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
```

```
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent) Handles
AxXMLGrid1.MouseMoveEvent
    With AxXMLGrid1
        Dim n As EXMLGRIDLib.Node = .get_NodeFromPoint(e.x, e.y)
        If Not n Is Nothing Then
            Debug.Print("You have clicked the '" & n.Name & "'.")
        End If
    End With
End Sub
```

The following C# sample prints the name of the node from the cursor:

```
private void axXMLGrid1_MouseMoveEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_MouseMoveEvent e)
{
    EXMLGRIDLib.Node node = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
    if (node != null)
        System.Diagnostics.Debug.Write(node.Name);
}
```

The following VFP sample prints the name of the node from the cursor:

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

with thisform.XMLGrid1
    n = .NodeFromPoint(x, y )
    if ( !isnull(n) )
        wait window nowait n.Name
    endif
endwith
```

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 DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the NodeFromPoint method to get the node from cursor.

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

C#

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

VB

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

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

C#

```
private void MouseUpEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,Button,Shift,X,Y)
RETURN
```

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

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

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

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

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

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

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

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

The following VB sample displays the node where the user releases the mouse:

```
Private Sub XMLGrid1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    With XMLGrid1
        Dim n As EXMLGRIDLibCtl.Node
        Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
```

```
    If Not n Is Nothing Then
      Debug.Print "You have clicked the '" & n.Name & "'."
    End If
  End With
End Sub
```

The following C++ sample displays the node where the user releases the mouse:

```
#include "Node.h"
void OnMouseUpXmlgrid1(short Button, short Shift, long X, long Y)
{
  CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
  if ( node.m_lpDispatch != NULL )
  {
    CString strName = node.GetName();
    OutputDebugString( strName );
  }
}
```

The following VB.NET sample displays the node where the user releases the mouse:

```
Private Sub AxXMLGrid1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_MouseUpEvent) Handles AxXMLGrid1.MouseUpEvent
  With AxXMLGrid1
    Dim n As EXMLGRIDLib.Node = .get_NodeFromPoint(e.x, e.y)
    If Not n Is Nothing Then
      Debug.Print("You have clicked the '" & n.Name & "'.")
    End If
  End With
End Sub
```

The following C# sample displays the node where the user releases the mouse:

```
private void axXMLGrid1_MouseUpEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_MouseUpEvent e)
{
  EXMLGRIDLib.Node node = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
  if (node != null)
    System.Diagnostics.Debug.Write(node.Name);
```

```
}
```

The following VFP sample displays the node where the user releases the mouse:

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

with thisform.XMLGrid1
   n = .NodeFromPoint(x, y )
   if ( !isnull(n) )
      wait window nowait n.Name
   endif
endwith
```

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. Use the OLEDropMode property to enable the OLE drag and drop operations in the control.

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,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_OLECompleteDragEvent);
begin
end;
```

Powe…

```
begin event OLECompleteDrag(long Effect)
end event OLECompleteDrag
```

VB.NET

```
Private Sub OLECompleteDrag(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,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 llEffect
    Forward Send OnComOLECompleteDrag llEffect
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 ExDataObject	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.

The OLEDragDrop event is fired when the user has dropped files or clipboard information into control. In order to enable OLE drag and drop feature into control you have to check the OLEDropMode property. The idea of drag and drop in the control is the same as in the other controls. To start accepting drag and drop sources the control should have the OLEDropMode property to exOLEDropManual. Once that is is set, the controls starts accepting any drag and drop sources.

Use the OLEDragDrop event to notify your application that user drags some data to the control. Use the Add method to insert new nodes to the control. Use the NodeFromPoint property to retrieve the node from the cursor. If the OLEDropMode property to exOLEDropManual and you need to drag data from the eXMLGrid control you need to handle the OLEStartDrag event. Use the Selected property to select a node. Use the EnsureVisibleNode method to ensure that a node fits the control's client area.

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#
```
private void OLEDragOver(object sender, AxEXMLGRIDLib._IXMLGridEvents_OLEDragOverEvent e)
{
}
```

C++
```
void OnOLEDragOver(LPDISPATCH Data,long FAR* Effect,short Button,short Shift,long X,long Y,short State)
{
}
```

C++ Builder
```
void __fastcall OLEDragOver(TObject *Sender,Exmlgridlib_tlb::IExDataObject *Data,long * Effect,short Button,short Shift,int X,int Y,short State)
{
}
```

Delphi
```
procedure OLEDragOver(ASender: TObject; Data : IExDataObject;var Effect :
```

```
Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer;State : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OLEDragOver(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_OLEDragOverEvent);
begin
end;
```

Powe…

```
begin event OLEDragOver(oleobject Data,long Effect,integer Button,integer
Shift,long X,long Y,integer State)
end event OLEDragOver
```

VB.NET

```
Private Sub OLEDragOver(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

VB6

```
Private Sub OLEDragOver(ByVal Data As EXMLGRIDLibCtl.IExDataObject,Effect As
Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As
Single,ByVal State As Integer)
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(oXMLGrid,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 llData Integer llEffect Short llButton Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY Short llState
    Forward Send OnComOLEDragOver llData llEffect llButton llShift llX llY llState
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.XMLGrid.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.XMLGrid.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.XMLGrid.1::OLE_YPOS_PIXELS,State as N)
end function
```

dBASE
```
function nativeObject_OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
return
```

The following VB sample adds a new node when user drags data to the control:

```
Private Sub XMLGrid1_OLEDragDrop(ByVal Data As EXMLGRIDLibCtl.IExDataObject, Effect As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
  With XMLGrid1
    Dim n As EXMLGRIDLibCtl.Node, nds As EXMLGRIDLibCtl.nodes
    Set nds = .nodes
    Set n = .NodeFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If Not n Is Nothing Then
      Set nds = n.nodes
    End If
    With nds
```

```
        Dim strData As String
        strData = Data.GetData(exCFText)
        .Add(strData).Selected = True
      End With
      If Not n Is Nothing Then
        n.Expanded = True
      End If
    End With
End Sub
```

The following C++ sample adds a new node when user drags data to the control:

```
#include "Node.h"
#import <exmlgrid.dll>
void OnOLEDragDropXmlgrid1(LPDISPATCH Data, long FAR* Effect, short Button, short
Shift, long X, long Y)
{
  COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
  if ( EXMLGRIDLib::IExDataObjectPtr spData = Data )
  {
    CString strData = V2S( &spData->GetData( EXMLGRIDLib::exCFText ) );
    CNodes nodes = m_xmlgrid.GetNodes();
    CNode node = m_xmlgrid.GetNodeFromPoint( X, Y );
    if ( node.m_lpDispatch != NULL )
      nodes = node.GetNodes();
    nodes.Add( strData, vtMissing, vtMissing ).SetSelected( TRUE );
    if ( node.m_lpDispatch != NULL )
      node.SetExpanded( TRUE );
  }
}
```

The #import <exmlgrid.dll> is called to import definitions for ExDataObject and ExDataObjectFiles objects. The #import <exmlgrid.dll> creates the EXMLGRIDLib namespace where all objects and types that eXMLGrid exports. If you need to drag data from eXMLGrid control to a window you need to use RegisterDragDrop API function. The RegisterDragDrop API function registers the specified window as one that can be the target of an OLE drag-and-drop operation and specifies the IDropTarget instance to use for drop operations. Shortly, you need an object that implements the IDropTarget interface, and to call the RegisterDragDrop API function.

The V2S function converts a VARIANT expression to a string expression:

```
static CString V2S( const VARIANT* pvtValue )
{
   COleVariant vtString;
   vtString.ChangeType( VT_BSTR, (VARIANT*)pvtValue );
   return V_BSTR( &vtString );
}
```

The following VB.NET sample adds a new node when user drags data to the control:

```
Private Sub AxXMLGrid1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_OLEDragDropEvent) Handles
AxXMLGrid1.OLEDragDrop
   With AxXMLGrid1
      Dim n As EXMLGRIDLib.Node = .get_NodeFromPoint(e.x, e.y), nds As
EXMLGRIDLib.Nodes = .Nodes
      If Not n Is Nothing Then
         nds = n.Nodes
      End If
      With nds
         Dim strData As String =
e.data.GetData(EXMLGRIDLib.exClipboardFormatEnum.exCFText)
         .Add(strData).Selected = True
      End With
      If Not n Is Nothing Then
         n.Expanded = True
      End If
   End With
End Sub
```

The following C# sample adds a new node when user drags data to the control:

```
private void axXMLGrid1_OLEDragDrop(object sender,
AxEXMLGRIDLib._IXMLGridEvents_OLEDragDropEvent e)
{
   EXMLGRIDLib.Nodes nodes = axXMLGrid1.Nodes;
   EXMLGRIDLib.Node n = axXMLGrid1.get_NodeFromPoint(e.x, e.y);
```

```
    if (n != null)
        nodes = n.Nodes;

nodes.Add(e.data.GetData(Convert.ToInt16(EXMLGRIDLib.exClipboardFormatEnum.exCFTex
 null, null).Selected = true;
    if (n != null)
        n.Expanded = true;
}
```

The following VFP sample adds a new node when user drags data to the control:

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

With thisform.XMLGrid1
    nds = .Nodes
    n = .NodeFromPoint(x , y)
    If !isnull(n) Then
        nds = n.Nodes
    EndIf
    With nds
        .Add(Data.GetData(1)).Selected = .t. && exCFText
    EndWith
    If !isnull(n) Then
        n.Expanded = .t.
    EndIf
EndWith
```

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 ExDataObject	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#
```
private void OLEDragOver(object sender,
AxEXMLGRIDLib._IXMLGridEvents_OLEDragOverEvent e)
{
}
```

C++
```
void OnOLEDragOver(LPDISPATCH Data,long FAR* Effect,short Button,short
Shift,long X,long Y,short State)
{
}
```

C++ Builder
```
void __fastcall OLEDragOver(TObject *Sender,Exmlgridlib_tlb::IExDataObject
*Data,long * Effect,short Button,short Shift,int X,int Y,short State)
{
}
```

Delphi
```
procedure OLEDragOver(ASender: TObject; Data : IExDataObject;var Effect :
Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer;State : Smallint);
begin
end;
```

Delphi 8 (.NET only)
```
procedure OLEDragOver(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_OLEDragOverEvent);
begin
end;
```

Powe…
```
begin event OLEDragOver(oleobject Data,long Effect,integer Button,integer
Shift,long X,long Y,integer State)
end event OLEDragOver
```

VB.NET
```
Private Sub OLEDragOver(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

VB6
```
Private Sub OLEDragOver(ByVal Data As EXMLGRIDLibCtl.IExDataObject,Effect As
Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As
Single,ByVal State As Integer)
```

```
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(oXMLGrid,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 llData Integer llEffect Short llButton Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY Short llState
    Forward Send OnComOLEDragOver llData llEffect llButton llShift llX llY llState
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.XMLGrid.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.XMLGrid.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.XMLGrid.1::OLE_YPOS_PIXELS,State as N)
```

```
end function
```

dBASE

```
function nativeObject_OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
return
```

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.

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,
AxEXMLGRIDLib._IXMLGridEvents_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:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_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(oXMLGrid,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 llEffect Boolean llDefaultCursors
    Forward Send OnComOLEGiveFeedback llEffect llDefaultCursors
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 ExDataObject	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, AxEXMLGRIDLib._IXMLGridEvents_OLESetDataEvent e)
{
}
```

C++
```
void OnOLESetData(LPDISPATCH Data,short Format)
{
}
```

C++ Builder
```
void __fastcall OLESetData(TObject *Sender,Exmlgridlib_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:
AxEXMLGRIDLib._IXMLGridEvents_OLESetDataEvent);
begin
end;
```

Powe…

```
begin event OLESetData(oleobject Data,integer Format)
end event OLESetData
```

VB.NET

```
Private Sub OLESetData(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_OLESetDataEvent) Handles OLESetData
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXMLGRIDLibCtl.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(oXMLGrid,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 llData Short llFormat
    Forward Send OnComOLESetData llData llFormat
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.XMLGrid.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 ExDataObject	An ExDataObject object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The programmer should provide the values for this parameter in this event. The SetData and Clear methods cannot be used here.
AllowedEffects as Long	A long containing the effects that the source component supports. The possible values are listed in Settings. The programmer should provide the values for this parameter in this event.

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 the Data object to provide the data that need to be dragged to other OLE component. Use the OLEDropMode property to enable the OLE drag and drop operations in the control.

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,
AxEXMLGRIDLib._IXMLGridEvents_OLEStartDragEvent e)
{
}
```

C++
```
void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
{
}
```

C++ Builder
```
void __fastcall OLEStartDrag(TObject *Sender,Exmlgridlib_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:
AxEXMLGRIDLib._IXMLGridEvents_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
AxEXMLGRIDLib._IXMLGridEvents_OLEStartDragEvent) Handles OLEStartDrag
End Sub
```

VB6
```
Private Sub OLEStartDrag(ByVal Data As
EXMLGRIDLibCtl.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(oXMLGrid,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 llData Integer llAllowedEffects
    Forward Send OnComOLEStartDrag llData llAllowedEffects
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.XMLGrid.1::IExDataObject,AllowedEffects as N)
end function
```

dBASE

```
function nativeObject_OLEStartDrag(Data,AllowedEffects)
return
```

The following samples shows how to enable drag and drop operation between exMLGrid control an other controls, using the OLEDropMode property on exOLEDropManual. If the **OLEDropMode property is exOLEDropManual** the OLEStartDrag and/or OLEDragDrop

events must be handled.

The following VB sample enables drag and drop nodes to a text editor:

```
Private Sub Form_Load()
   With XMLGrid1
      .BeginUpdate
         .OLEDropMode = exOLEDropManual
         With .Editors.Add("Float", EditType)
            .Numeric = exFloat
         End With
         With .Editors.Add("DropDown", DropDownListType)
            .AddItem 1, "Yes"
            .AddItem 2, "No"
         End With
         With .Nodes
            With .Add("Root").Nodes
               With .Add("Child 1", "1.2")
                  .Editor = "Float"
               End With
               With .Add("Child 2", "1")
                  .Editor = "DropDown"
               End With
            End With
         End With
      .EndUpdate
   End With
End Sub

Private Sub XMLGrid1_OLEStartDrag(ByVal Data As EXMLGRIDLibCtl.IExDataObject,
AllowedEffects As Long)
   AllowedEffects = EXMLGRIDLibCtl.exOLEDropEffectCopy
   Data.SetData XMLGrid1.FocusNode.Name, EXMLGRIDLibCtl.exCFText
End Sub
```

The following C++ sample enables drag and drop nodes to a text editor:

```
#include "Node.h"
```

```
#import <exmlgrid.dll>
void OnOLEStartDragXmlgrid1(LPDISPATCH Data, long FAR* AllowedEffects)
{
   *AllowedEffects = EXMLGRIDLib::exOLEDropEffectCopy;
   if ( EXMLGRIDLib::IExDataObjectPtr spData = Data )
   {
      COleVariant vtValue = m_xmlgrid.GetFocusNode().GetName();
      spData->SetData( vtValue, COleVariant( (long)EXMLGRIDLib::exCFText ) );
   }
}
```

The #import <exmlgrid.dll> is called to import definitions for ExDataObject and ExDataObjectFiles objects. The #import <exmlgrid.dll> creates the EXMLGRIDLib namespace where all objects and types that eXMLGrid exports. If you need to drag data from eXMLGrid control to a window you need to use RegisterDragDrop API function. The RegisterDragDrop API function registers the specified window as one that can be the target of an OLE drag-and-drop operation and specifies the IDropTarget instance to use for drop operations. Shortly, you need an object that implements the IDropTarget interface, and to call the RegisterDragDrop API function. The OLEStartDrag event is not called if the OLEDropMode property is exOLEDropNone.

The following VB.NET sample enables drag and drop nodes to a text editor:

```
Private Sub AxXMLGrid1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_OLEStartDragEvent) Handles AxXMLGrid1.OLEStartDrag
   e.allowedEffects = EXMLGRIDLib.exOLEDropEffectEnum.exOLEDropEffectCopy
   e.data.SetData(AxXMLGrid1.FocusNode.Name,
EXMLGRIDLib.exClipboardFormatEnum.exCFText)
End Sub
```

The OLEStartDrag event is not called if the OLEDropMode property is exOLEDropNone.

The following C# sample enables drag and drop nodes to a text editor:

```
private void axXMLGrid1_OLEStartDrag(object sender,
AxEXMLGRIDLib._IXMLGridEvents_OLEStartDragEvent e)
{
   e.allowedEffects =
Convert.ToUInt16(EXMLGRIDLib.exOLEDropEffectEnum.exOLEDropEffectCopy);
   e.data.SetData(axXMLGrid1.FocusNode.Name,
```

```
EXMLGRIDLib.exClipboardFormatEnum.exCFText);
}
```

The OLEStartDrag event is not called if the OLEDropMode property is exOLEDropNone.

The following VFP sample enables drag and drop nodes to a text editor:

```
*** ActiveX Control Event ***
LPARAMETERS data, allowedeffects

allowedeffects = 1 && exOLEDropEffectCopy
with thisform.XMLGrid1
   data.SetData( .FocusNode.Name, 1 )  && exCFText
endwith
```

The OLEStartDrag event is not called if the OLEDropMode property is exOLEDropNone.

event RemoveNode (Node as Node)

Occurs when a node is removed from the nodes collection.

Type	Description
Node as Node	A Node object being removed.

The RemoveNode event notifies your application that a node is removed. Use the RemoveNode event to remove any extra data that you have associated to a node. Use the Remove method to remove a node. Use the Clear method to clear the nodes collection. Use the Nodes property to access the control's nodes collection. Use the Nodes property to access the node's child nodes collection. Use the UserData property to assign an extra data to a node.

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

C#
```
private void RemoveNode(object sender,exontrol.EXMLGRIDLib.Node Node)
{
}
```

VB
```
Private Sub RemoveNode(ByVal sender As System.Object,ByVal Node As
exontrol.EXMLGRIDLib.Node) Handles RemoveNode
End Sub
```

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

C#
```
private void RemoveNode(object sender,
AxEXMLGRIDLib._IXMLGridEvents_RemoveNodeEvent e)
{
}
```

C++
```
void OnRemoveNode(LPDISPATCH Node)
{
}
```

C++ Builder
```
void __fastcall RemoveNode(TObject *Sender,Exmlgridlib_tlb::INode *Node)
{
}
```

Delphi
```
procedure RemoveNode(ASender: TObject; Node : INode);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure RemoveNode(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_RemoveNodeEvent);
begin
end;
```

Powe…

```
begin event RemoveNode(oleobject Node)
end event RemoveNode
```

VB.NET

```
Private Sub RemoveNode(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_RemoveNodeEvent) Handles RemoveNode
End Sub
```

VB6

```
Private Sub RemoveNode(ByVal Node As EXMLGRIDLibCtl.INode)
End Sub
```

VBA

```
Private Sub RemoveNode(ByVal Node As Object)
End Sub
```

VFP

```
LPARAMETERS Node

```

Xbas…

```
PROCEDURE OnRemoveNode(oXMLGrid,Node)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComRemoveNode Variant llNode
   Forward Send OnComRemoveNode llNode
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_RemoveNode(Node) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_RemoveNode(COM _Node)
{
}
```

XBasic

```
function RemoveNode as v (Node as OLE::Exontrol.XMLGrid.1::INode)
end function
```

dBASE

```
function nativeObject_RemoveNode(Node)
return
```

The following VB sample displays the name of the node being removed:

```
Private Sub XMLGrid1_RemoveNode(ByVal Node As EXMLGRIDLibCtl.INode)
    Debug.Print Node.Name
End Sub
```

The following C++ sample displays the name of the node being removed:

```
#include "Node.h"
void OnRemoveNodeXmlgrid1(LPDISPATCH Node)
{
    CNode node( Node ); node.m_bAutoRelease = FALSE;
    CString strName = node.GetName();
    OutputDebugString( strName );
}
```

The following VB.NET sample displays the name of the node being removed:

```
Private Sub AxXMLGrid1_RemoveNode(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_RemoveNodeEvent) Handles AxXMLGrid1.RemoveNode
    Debug.Print(e.node.Name)
End Sub
```

The following C# sample displays the name of the node being removed:

```
private void axXMLGrid1_RemoveNode(object sender,
AxEXMLGRIDLib._IXMLGridEvents_RemoveNodeEvent e)
{
    System.Diagnostics.Debug.Write(e.node.Name);
}
```

The following VFP sample displays the name of the node being removed:

```
*** ActiveX Control Event ***
LPARAMETERS node

wait window nowait node.Name
```

event ResizeLevel (Level as Long)

Occurs when the user resizes the level.

Type	Description
Level as Long	A long expression that indicates the level being resized. The Level parameter is zero based. The 0 Level indicates the first level. The 1 Level indicates the second level and so on.

Use the ResizeLevel event to notify your application when user resizes a level. Use the LevelWidth property to specify the level's width. Use the Level property to get the node's level.

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

C#

```
private void ResizeLevel(object sender,int Level)
{
}
```

VB

```
Private Sub ResizeLevel(ByVal sender As System.Object,ByVal Level As Integer)
Handles ResizeLevel
End Sub
```

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

C#

```
private void ResizeLevel(object sender,
AxEXMLGRIDLib._IXMLGridEvents_ResizeLevelEvent e)
{
}
```

C++

```
void OnResizeLevel(long Level)
{
}
```

C++ Builder

```
void __fastcall ResizeLevel(TObject *Sender,long Level)
{
}
```

Delphi

```
procedure ResizeLevel(ASender: TObject; Level : Integer);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure ResizeLevel(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_ResizeLevelEvent);
begin
end;
```

Powe…

```
begin event ResizeLevel(long Level)
end event ResizeLevel
```

VB.NET

```
Private Sub ResizeLevel(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_ResizeLevelEvent) Handles ResizeLevel
End Sub
```

VB6

```
Private Sub ResizeLevel(ByVal Level As Long)
End Sub
```

VBA

```
Private Sub ResizeLevel(ByVal Level As Long)
End Sub
```

VFP

```
LPARAMETERS Level

```

Xbas…

```
PROCEDURE OnResizeLevel(oXMLGrid,Level)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComResizeLevel Integer llLevel
    Forward Send OnComResizeLevel llLevel
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_ResizeLevel(Level) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ResizeLevel(int _Level)
{
}
```

XBasic

```
function ResizeLevel as v (Level as N)
end function
```

dBASE

```
function nativeObject_ResizeLevel(Level)
return
```

The following VB sample specifies a minimum width for the first level:

```
Private Sub XMLGrid1_ResizeLevel(ByVal Level As Long)
   If Level = 0 Then
      With XMLGrid1
         If .LevelWidth(Level) < 64 Then
            .LevelWidth(Level) = 64
         End If
      End With
   End If
End Sub
```

The following C++ sample specifies a minimum width for the first level:

```
void OnResizeLevelXmlgrid1(long Level)
{
   if ( Level == 0 )
      if ( m_xmlgrid.GetLevelWidth( Level ) < 64 )
         m_xmlgrid.SetLevelWidth( Level, 64 );
}
```

The following VB.NET sample specifies a minimum width for the first level:

```
Private Sub AxXMLGrid1_ResizeLevel(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_ResizeLevelEvent) Handles AxXMLGrid1.ResizeLevel
   If (e.level = 0) Then
      With AxXMLGrid1
         If (.get_LevelWidth(e.level) < 64) Then
            .set_LevelWidth(e.level, 64)
         End If
      End With
   End If
End Sub
```

The following C# sample specifies a minimum width for the first level:

```
private void axXMLGrid1_ResizeLevel(object sender,
AxEXMLGRIDLib._IXMLGridEvents_ResizeLevelEvent e)
{
   if (e.level == 0)
      if (axXMLGrid1.get_LevelWidth(e.level) < 64)
         axXMLGrid1.set_LevelWidth(e.level, 64);
}
```

The following VFP sample specifies a minimum width for the first level:

```
*** ActiveX Control Event ***
LPARAMETERS level

with thisform.XMLGrid1
   if ( level = 0 )
      if ( .LevelWidth(level) < 64 )
         .LevelWidth(level) = 64
      endif
   endif
endwith
```

event ScrollButtonClick (ScrollBar as ScrollBarEnum, ScrollPart as ScrollPartEnum)

Occurs when the user clicks a button in the scrollbar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that specifies the scroll bar being clicked.
ScrollPart as ScrollPartEnum	A ScrollPartEnum expression that indicates the part of the scroll being clicked.

Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar. The ScrollButtonClick event is fired when the user clicks and releases the mouse over an enabled part of the scroll bar. Use the ScrollBars property to specify the visible scrollbars in the control. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the Background property to change the visual appearance for any part in the control's scroll bar.

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

C#

```
private void ScrollButtonClick(object sender,exontrol.EXMLGRIDLib.ScrollBarEnum ScrollBar,exontrol.EXMLGRIDLib.ScrollPartEnum ScrollPart)
{
}
```

VB

```
Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As exontrol.EXMLGRIDLib.ScrollBarEnum,ByVal ScrollPart As exontrol.EXMLGRIDLib.ScrollPartEnum) Handles ScrollButtonClick
End Sub
```

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

C#

```
private void ScrollButtonClick(object sender,
AxEXMLGRIDLib._IXMLGridEvents_ScrollButtonClickEvent e)
{
}
```

C++

```
void OnScrollButtonClick(long ScrollBar,long ScrollPart)
{
```

```
}
```

C++ Builder

```
void __fastcall ScrollButtonClick(TObject *Sender,Exmlgridlib_tlb::ScrollBarEnum ScrollBar,Exmlgridlib_tlb::ScrollPartEnum ScrollPart)
{
}
```

Delphi

```
procedure ScrollButtonClick(ASender: TObject; ScrollBar : ScrollBarEnum;ScrollPart : ScrollPartEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ScrollButtonClick(sender: System.Object; e: AxEXMLGRIDLib._IXMLGridEvents_ScrollButtonClickEvent);
begin
end;
```

Powe…

```
begin event ScrollButtonClick(long ScrollBar,long ScrollPart)
end event ScrollButtonClick
```

VB.NET

```
Private Sub ScrollButtonClick(ByVal sender As System.Object, ByVal e As AxEXMLGRIDLib._IXMLGridEvents_ScrollButtonClickEvent) Handles ScrollButtonClick
End Sub
```

VB6

```
Private Sub ScrollButtonClick(ByVal ScrollBar As EXMLGRIDLibCtl.ScrollBarEnum,ByVal ScrollPart As EXMLGRIDLibCtl.ScrollPartEnum)
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart

```

Xbas…

```
PROCEDURE OnScrollButtonClick(oXMLGrid,ScrollBar,ScrollPart)
RETURN
```

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

Java…
```
<SCRIPT EVENT="ScrollButtonClick(ScrollBar,ScrollPart)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function ScrollButtonClick(ScrollBar,ScrollPart)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComScrollButtonClick OLEScrollBarEnum llScrollBar
OLEScrollPartEnum llScrollPart
    Forward Send OnComScrollButtonClick llScrollBar llScrollPart
End_Procedure
```

Visual Objects
```
METHOD OCX_ScrollButtonClick(ScrollBar,ScrollPart) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_ScrollButtonClick(int _ScrollBar,int _ScrollPart)
{
}
```

XBasic
```
function ScrollButtonClick as v (ScrollBar as
OLE::Exontrol.XMLGrid.1::ScrollBarEnum,ScrollPart as
OLE::Exontrol.XMLGrid.1::ScrollPartEnum)
end function
```

dBASE
```
function nativeObject_ScrollButtonClick(ScrollBar,ScrollPart)
return
```

The following VB sample displays the identifier of the scroll's button being clicked:

```
With XMLGrid1
   .BeginUpdate
      .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
      .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
      .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
```

```
   .EndUpdate
End With
```

```
Private Sub XMLGrid1_ScrollButtonClick(ByVal ScrollPart As
EXXMLGRIDLibCtl.ScrollPartEnum)
   MsgBox (ScrollPart)
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxXMLGrid1
   .BeginUpdate()
   .set_ScrollPartVisible(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part Or
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2")
   .EndUpdate()
End With
```

```
Private Sub AxXMLGrid1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As
AxEXXMLGRIDLib._IXMLGridEvents_ScrollButtonClickEvent) Handles
AxXMLGrid1.ScrollButtonClick
      MessageBox.Show( e.scrollPart.ToString())
End Sub
```

The following C# sample displays the identifier of the scroll's button being clicked:

```
axXMLGrid1.BeginUpdate();
axXMLGrid1.set_ScrollPartVisible(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part |
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, true);
axXMLGrid1.set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axXMLGrid1.set_ScrollPartCaption(EXXMLGRIDLib.ScrollBarEnum.exVScroll,
EXXMLGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
```

```
axXMLGrid1.EndUpdate();
```

```
private void axXMLGrid1_ScrollButtonClick(object sender,
AxEXXMLGRIDLib._IXMLGridEvents_ScrollButtonClickEvent e)
{
   MessageBox.Show(e.scrollPart.ToString());
}
```

The following C++ sample displays the identifier of the scroll's button being clicked:

```
m_xmlGrid.BeginUpdate();
m_xmlGrid.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_xmlGrid.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_xmlGrid.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img>
</img>2") );
m_xmlGrid.EndUpdate();
```

```
void OnScrollButtonClickXMLGrid1(long ScrollPart)
{
   CString strFormat;
   strFormat.Format( _T("%i"), ScrollPart );
   MessageBox( strFormat );
}
```

The following VFP sample displays the identifier of the scroll's button being clicked:

```
With thisform.XMLGrid1
   .BeginUpdate
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWith
```

```
*** ActiveX Control Event ***
LPARAMETERS scrollpart
```

```
wait window nowait ltrim(str(scrollpart))
```

event SelectionChanged ()

Fires when the user changes the selection.

Type	Description

Use the SelectionChanged event to notify your application that the user changes the selection. Use the SingleSel property to specify whether the control supports single or multiple selection. Use the FocusNode property to retrieve the focused node. Use the SelectCount property to get the number of selected nodes. Use the SelectedNode property to retrieve the selected node giving its index in the selected nodes collection. Use the Selected property to select a node. Use the SelForeColor, SelForeColorChild, SelBackColor, SelBackColorChild properties to customize the colors for selected nodes.

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(oXMLGrid)
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 displays the selected node(s), as soon as the user changes the selection:

```
Private Sub XMLGrid1_SelectionChanged()
   With XMLGrid1
      Dim i As Long
      For i = 0 To .SelectCount - 1
         Debug.Print .SelectedNode(i).Name
      Next
   End With
End Sub
```

The following C++ sample displays the selected node(s), as soon as the user changes the selection:

```
#include "Node.h"
void OnSelectionChangedXmlgrid1()
{
   if ( IsWindow( m_xmlgrid.m_hWnd ) )
      for ( long i = 0; i < m_xmlgrid.GetSelectCount(); i++ )
      {
         CNode node = m_xmlgrid.GetSelectedNode( COleVariant( i ) );
         OutputDebugString( node.GetName() );
```

```
    }
}
```

The following VB.NET sample displays the selected node(s), as soon as the user changes the selection:

```
Private Sub AxXMLGrid1_SelectionChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxXMLGrid1.SelectionChanged
  With AxXMLGrid1
    Dim i As Long
    For i = 0 To .SelectCount - 1
      Debug.Write(.get_SelectedNode(i).Name())
    Next
  End With
End Sub
```

The following C# sample displays the selected node(s), as soon as the user changes the selection:

```
private void axXMLGrid1_SelectionChanged(object sender, EventArgs e)
{
  for (int i = 0; i < axXMLGrid1.SelectCount; i++)
  {
    EXMLGRIDLib.Node node = axXMLGrid1.get_SelectedNode(i);
    System.Diagnostics.Debug.Write(node.Name);
  }
}
```

The following VFP sample displays the selected node(s), as soon as the user changes the selection:

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

With thisform.XMLGrid1
  local i
  For i = 0 To .SelectCount - 1
    wait window nowait .SelectedNode(i).Name
  Next
EndWith
```

event UserEditorClose (Object as Object, Node as Node)

Fired the user editor is about to be opened.

Type	Description
Object as Object	An object created by UserEditor property
Node as Node	A Node object where the ActiveX editor is closed.

Use the UserEditorClose event to notify your application when the user editor is hidden. The control fires UserEditorOleEvent event each time when a an user editor object fires an event. Use the Add method and UserEditorType type to add an ActiveX editor to the control. Use the UserEditor method to create an ActiveX editor. Use the UserEditorObject property to get the ActiveX editor created by the UserEditor method. The UserEditorOpen event is fired when the control shows an ActiveX editor.

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

C#
```
private void UserEditorClose(object sender,object
Obj,exontrol.EXMLGRIDLib.Node Node)
{
}
```

VB
```
Private Sub UserEditorClose(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Node As exontrol.EXMLGRIDLib.Node) Handles UserEditorClose
End Sub
```

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

C#
```
private void UserEditorClose(object sender,
AxEXMLGRIDLib._IXMLGridEvents_UserEditorCloseEvent e)
{
}
```

C++
```
void OnUserEditorClose(LPDISPATCH Object,LPDISPATCH Node)
{
}
```

C++ Builder
```
void __fastcall UserEditorClose(TObject *Sender,IDispatch
*Object,Exmlgridlib_tlb::INode *Node)
{
```

```
}
```

Delphi

```
procedure UserEditorClose(ASender: TObject; Object : IDispatch;Node : INode);
begin
end;
```

Delphi 8 (.NET only)

```
procedure UserEditorClose(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_UserEditorCloseEvent);
begin
end;
```

Powe…

```
begin event UserEditorClose(oleobject Object,oleobject Node)
end event UserEditorClose
```

VB.NET

```
Private Sub UserEditorClose(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_UserEditorCloseEvent) Handles UserEditorClose
End Sub
```

VB6

```
Private Sub UserEditorClose(ByVal Object As Object,ByVal Node As
EXMLGRIDLibCtl.INode)
End Sub
```

VBA

```
Private Sub UserEditorClose(ByVal Object As Object,ByVal Node As Object)
End Sub
```

VFP

```
LPARAMETERS Object,Node

```

Xbas…

```
PROCEDURE OnUserEditorClose(oXMLGrid,Object,Node)
RETURN
```

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

Java…

```
<SCRIPT EVENT="UserEditorClose(Object,Node)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function UserEditorClose(Object,Node)
```

```
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComUserEditorClose Variant llObject Variant llNode
    Forward Send OnComUserEditorClose llObject llNode
End_Procedure
```

Visual Objects

```
METHOD OCX_UserEditorClose(Object,Node) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_UserEditorClose(COM _Object,COM _Node)
{
}
```

XBasic

```
function UserEditorClose as v (Object as P,Node as
OLE::Exontrol.XMLGrid.1::INode)
end function
```

dBASE

```
function nativeObject_UserEditorClose(Object,Node)
return
```

The following VB sample changes the node's Value property when the user editor is closed (in this case we used the Exontrol's ExComboBox control):

```
Private Sub XMLGrid1_UserEditorClose(ByVal Object As Object, ByVal Node As
EXMLGRIDLibCtl.INode)
On Error Resume Next
   With Object.Items
      Node.Value = .Select(0)
   End With
End Sub
```

The following C++ sample changes the node's Value property when the user editor is closed (in this case we have used the Exontrol's ExComboBox component):

```
#import <excombobox.dll>
void OnUserEditorCloseXmlgrid1(LPDISPATCH Object, LPDISPATCH Node)
{
```

```
    COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
    CNode node( Node ); node.m_bAutoRelease = FALSE;
    EXCOMBOBOXLib::IComboBoxPtr spComboBox = Object;
    if ( spComboBox != NULL )
    {
        COleVariant vtValue;
        if ( SUCCEEDED( spComboBox->get_Select( COleVariant( (long)0 ), &vtValue  ) ) )
            node.SetValue( vtValue );
    }
}
```

The sample assumes that the Object parameter holds an ExComboBox control. We need to call the #import <excombobox.dll> in order to include definitions for objects and types in the ExComboBox control. The #import <excombobox.dll> creates EXCOMBOBOXLib namespace that includes all definitions for objects and types that the ExComboBox control exports.

The following VB.NET sample changes the node's Value property when the user editor is closed (in this case we have used the Exontrol's ExComboBox component):

```
Private Sub AxXMLGrid1_UserEditorClose(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_UserEditorCloseEvent) Handles
AxXMLGrid1.UserEditorClose
    On Error Resume Next
    With e.object.Items
        e.node.Value = .Select(0)
    End With
End Sub
```

The following C# sample changes the node's Value property when the user editor is closed (in this case we have used the Exontrol's ExComboBox component):

```
private void axXMLGrid1_UserEditorClose(object sender,
AxEXMLGRIDLib._IXMLGridEvents_UserEditorCloseEvent e)
{
    EXCOMBOBOXLib.ComboBox comboBox = e.@object as EXCOMBOBOXLib.ComboBox;
    if (comboBox != null)
        e.node.Value = comboBox.get_Select(0);
```

```
}
```

In C# your project needs a new reference to the Exontrol's ExComboBox control library. Use the Project\Add Reference\COM item to add new reference to a COM object. Once that you added a reference to the Exontrol's ExComboBox the EXCOMBOBOXLib namespace is created. The EXCOMBOBOXLib namespace contains definitions for all objects that ExComboBox control exports.

The following VFP sample changes the node's Value property when the user editor is closed (in this case we have used the Exontrol's ExComboBox component):

```
*** ActiveX Control Event ***
LPARAMETERS object, node


node.value = object.Select(0)
```

event UserEditorOleEvent (Object as Object, Ev as OleEvent, CloseEditor as Boolean, Node as Node)

Occurs when an user editor fires an event.

Type	Description
Object as Object	An object created by UserEditor property.
Ev as OleEvent	An OleEvent object that holds information about the event
CloseEditor as Boolean	A boolean expression that indicates whether the control should close the user editor.
Node as Node	A Node object where the ActiveX editor is opened.

The UserEditorOleEvent is fired every time when an user editor object fires an event. The information about fired event is stored by Ev parameter. The CloseEditor parameter is useful to inform the control when the editor should be hidden. The UserEditorOpen event is fired when the control shows an ActiveX editor. The control fires the UserEditorClose event when the user closes the ActiveX editor (for instance, when he clicks outside the editing node). Use the Add method and UserEditorType type to add an ActiveX editor to the control. Use the UserEditor method to create an ActiveX editor. Use the UserEditorObject property to get the ActiveX editor created by the UserEditor method.

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

C#
```
private void UserEditorOleEvent(object sender,object
Obj,exontrol.EXMLGRIDLib.OleEvent Ev,ref bool
CloseEditor,exontrol.EXMLGRIDLib.Node Node)
{
}
```

VB
```
Private Sub UserEditorOleEvent(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Ev As exontrol.EXMLGRIDLib.OleEvent,ByRef CloseEditor As
Boolean,ByVal Node As exontrol.EXMLGRIDLib.Node) Handles UserEditorOleEvent
End Sub
```

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

C#
```
private void UserEditorOleEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOleEventEvent e)
{
}
```

C++

```
void OnUserEditorOleEvent(LPDISPATCH Object,LPDISPATCH Ev,BOOL FAR*
CloseEditor,LPDISPATCH Node)
{
}
```

C++ Builder

```
void __fastcall UserEditorOleEvent(TObject *Sender,IDispatch
*Object,Exmlgridlib_tlb::IOleEvent *Ev,VARIANT_BOOL *
CloseEditor,Exmlgridlib_tlb::INode *Node)
{
}
```

Delphi

```
procedure UserEditorOleEvent(ASender: TObject; Object : IDispatch;Ev :
IOleEvent;var CloseEditor : WordBool;Node : INode);
begin
end;
```

Delphi 8 (.NET only)

```
procedure UserEditorOleEvent(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOleEventEvent);
begin
end;
```

Powe…

```
begin event UserEditorOleEvent(oleobject Object,oleobject Ev,boolean
CloseEditor,oleobject Node)
end event UserEditorOleEvent
```

VB.NET

```
Private Sub UserEditorOleEvent(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOleEventEvent) Handles
UserEditorOleEvent
End Sub
```

VB6

```
Private Sub UserEditorOleEvent(ByVal Object As Object,ByVal Ev As
EXMLGRIDLibCtl.IOleEvent,CloseEditor As Boolean,ByVal Node As
EXMLGRIDLibCtl.INode)
End Sub
```

VBA

```
Private Sub UserEditorOleEvent(ByVal Object As Object,ByVal Ev As
Object,CloseEditor As Boolean,ByVal Node As Object)
End Sub
```

VFP

```
LPARAMETERS Object,Ev,CloseEditor,Node
```

Xbas…

```
PROCEDURE OnUserEditorOleEvent(oXMLGrid,Object,Ev,CloseEditor,Node)
RETURN
```

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

Java…

```
<SCRIPT EVENT="UserEditorOleEvent(Object,Ev,CloseEditor,Node)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function UserEditorOleEvent(Object,Ev,CloseEditor,Node)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComUserEditorOleEvent Variant llObject Variant llEv Boolean
llCloseEditor Variant llNode
    Forward Send OnComUserEditorOleEvent llObject llEv llCloseEditor llNode
End_Procedure
```

Visual Objects

```
METHOD OCX_UserEditorOleEvent(Object,Ev,CloseEditor,Node) CLASS
MainDialog
RETURN NIL
```

X++

```
void onEvent_UserEditorOleEvent(COM _Object,COM _Ev,COMVariant /*bool*/
_CloseEditor,COM _Node)
{
}
```

XBasic

```
function UserEditorOleEvent as v (Object as P,Ev as
OLE::Exontrol.XMLGrid.1::IOleEvent,CloseEditor as L,Node as
OLE::Exontrol.XMLGrid.1::INode)
end function
```

dBASE

```
function nativeObject_UserEditorOleEvent(Object,Ev,CloseEditor,Node)
return
```

The following VB sample closes the Exontrol's ExComboBox user editor when the user selects a new value, or when it presses the Escape key. Also the sample changes the value of the node in the ExComboBox control:

```
Private Sub XMLGrid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXMLGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Node As
EXMLGRIDLibCtl.INode)
   If (Ev.Name = "Change") Then
      Node.Value = Object.Select(0)
      CloseEditor = True
   End If

   If (Ev.Name = "KeyPress") Then
      Dim l As Long
      l = Ev(0).Value
      If l = vbKeyEscape Then
         CloseEditor = True
      End If
   End If
End Sub
```

the sample requires an ActiveX editor (in our case the Exontrol's ExComboBox control):

```
With XMLGrid1.Editors
   With .Add("excombobox", UserEditorType)
      .UserEditor "Exontrol.ComboBox", ""
      With .UserEditorObject
         .BeginUpdate
         .LabelHeight = XMLGrid1.NodeHeight - 3
         .LinesAtRoot = True
         .HeightList = 256
         .WidthList = 256
         .IntegralHeight = True
         .Columns.Add ("Name")
         .Columns.Add ("Value")
         .ColumnAutoResize = True
         With .Items
```

```
            Dim h As Long, h1 As Long
            h = .AddItem("Item 1")
            .CellCaption(h, 1) = "Item 1.2"
            h1 = .InsertItem(h, , "SubItem 1")
            .CellCaption(h1, 1) = "SubItem 1.2"
            h1 = .InsertItem(h, , "SubItem 2")
            .CellCaption(h1, 1) = "SubItem 2.2"
            .ExpandItem(h) = True
          End With
          .EndUpdate
        End With
    End With
End With
```

The following C++ sample closes the Exontrol's ExComboBox user editor when the user selects a new value, or when it presses the Escape key. Also the sample changes the value of the node in the ExComboBox control:

```
#import <exmlgrid.dll>
#import <excombobox.dll>
void OnUserEditorOleEventXmlgrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, LPDISPATCH Node)
{
    EXMLGRIDLib::IOleEventPtr spEvent = Ev;
    EXCOMBOBOXLib::IComboBoxPtr spComboBox = Object;
    if ( spComboBox != NULL )
        if ( spEvent != NULL  )
        {
            if ( spEvent->Name.operator ==( "Change" ) )
            {
                CNode node( Node );
                node.SetValue( spComboBox->GetSelect( COleVariant( (long)0 ) ) );
                *CloseEditor = TRUE;
            }
            else
                if ( spEvent->Name.operator ==( "KeyPress" ) )
                {
                    // gets the KeyCode parameter
```

```
            EXMLGRIDLib::IOleEventParamPtr spParam;
            if ( SUCCEEDED( spEvent->get_Param( COleVariant( (long)0 ), &spParam ) ) )
            {
               COleVariant vtI4;
               vtI4.ChangeType( VT_I4, &spParam->Value );
               if ( V_I4( &vtI4) == VK_ESCAPE )
                  *CloseEditor = TRUE;
            }
         }
      }
}
```

the sample requires an ActiveX editor (in our case the Exontrol's ExComboBox control), so we need to call the #import <excombobox.dll> in order to include definitions for objects and types in the ExComboBox control. The #import <excombobox.dll> creates EXCOMBOBOXLib namespace that includes all definitions for objects and types that the ExComboBox control exports.

```
#include "Editor.h"
#include "Editors.h"
COleVariant vtMissing; V_VT( &vtMissing; ) = VT_ERROR;
CEditors editors = m_xmlgrid.GetEditors();
CEditor editor = editors.Add( COleVariant( "excombobox" ), 16 /*UserEditorType*/ );
editor.UserEditor( "Exontrol.ComboBox", "" );
EXCOMBOBOXLib::IComboBoxPtr spComboBox = editor.GetUserEditorObject();
if ( spComboBox != NULL )
{
   spComboBox->BeginUpdate();
   spComboBox->LabelHeight = m_xmlgrid.GetNodeHeight() - 3;
   spComboBox->LinesAtRoot = EXCOMBOBOXLib::exLinesAtRoot;
   spComboBox->put_HeightList( vtMissing, 256 );
   spComboBox->put_WidthList( vtMissing, 256 );
   spComboBox->IntegralHeight = true;
   spComboBox->Columns->Add("Name");
   spComboBox->Columns->Add("Value");
   spComboBox->ColumnAutoResize = true;
   EXCOMBOBOXLib::IItemsPtr spItems = spComboBox->Items;
   long h = spItems->AddItem(COleVariant( "Item 1" ));
```

```
    spItems->put_CellCaption(COleVariant(h),COleVariant((long)1), COleVariant( "Item 1.2" )
);
    long h1 = spItems->InsertItem(h, vtMissing, COleVariant( "SubItem 1") );
    spItems->put_CellCaption(COleVariant(h1),COleVariant((long)1), COleVariant( "SubItem
1.2" ) );
    h1 = spItems->InsertItem(h, vtMissing, COleVariant( "SubItem 2") );
    spItems->put_CellCaption(COleVariant(h1),COleVariant((long)1), COleVariant( "SubItem
2.2" ) );
    spItems->put_ExpandItem(h, true );
    spComboBox->EndUpdate();
}
```

The following VB.NET sample closes the Exontrol's ExComboBox user editor when the user selects a new value, or when it presses the Escape key. Also the sample changes the value of the node in the ExComboBox control:

```
Private Sub AxXMLGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOleEventEvent) Handles
AxXMLGrid1.UserEditorOleEvent
    If (e.ev.Name = "Change") Then
        e.node.Value = e.object.Select(0)
        e.closeEditor = True
    End If

    If (e.ev.Name = "KeyPress") Then
        Dim l As Integer = e.ev(0).Value
        If l = Keys.Escape Then
            e.closeEditor = True
        End If
    End If
End Sub
```

the sample requires an ActiveX editor (in our case the Exontrol's ExComboBox control):

```
With AxXMLGrid1.Editors
    With .Add("excombobox", EXMLGRIDLib.EditTypeEnum.UserEditorType)
        .UserEditor("Exontrol.ComboBox", "")
        With .UserEditorObject
```

```
        .BeginUpdate()
        .LabelHeight = AxXMLGrid1.NodeHeight - 3
        .LinesAtRoot = True
        .HeightList = 256
        .WidthList = 256
        .IntegralHeight = True
        .Columns.Add("Name")
        .Columns.Add("Value")
        .ColumnAutoResize = True
        With .Items
            Dim h, h1 As Integer
            h = .AddItem("Item 1")
            .CellCaption(h, 1) = "Item 1.2"
            h1 = .InsertItem(h, , "SubItem 1")
            .CellCaption(h1, 1) = "SubItem 1.2"
            h1 = .InsertItem(h, , "SubItem 2")
            .CellCaption(h1, 1) = "SubItem 2.2"
            .ExpandItem(h) = True
        End With
        .EndUpdate()
    End With
  End With
End With
```

The following C# sample closes the Exontrol's ExComboBox user editor when the user selects a new value, or when it presses the Escape key. Also the sample changes the value of the node in the ExComboBox control:

```
private void axXMLGrid1_UserEditorOleEvent(object sender,
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOleEventEvent e)
{
  if (e.ev.Name == "Change")
  {
    EXCOMBOBOXLib.ComboBox comboBox = e.@object as
EXCOMBOBOXLib.ComboBox;
    if ( comboBox != null )
      e.node.Value = comboBox.get_Select(0);
    e.closeEditor = true;
```

```
    }
    else
        if (e.ev.Name == "KeyPress")
        {
            if (Convert.ToInt16(e.ev[0].Value) == Convert.ToInt16(Keys.Escape))
                e.closeEditor = true;
        }
}
```

the sample requires an ActiveX editor (in our case the Exontrol's ExComboBox control):

```
EXMLGRIDLib.Editor editor = axXMLGrid1.Editors.Add("excombobox",
EXMLGRIDLib.EditTypeEnum.UserEditorType);
editor.UserEditor("Exontrol.ComboBox", "");
EXCOMBOBOXLib.ComboBox comboBox = editor.UserEditorObject as
EXCOMBOBOXLib.ComboBox;
if ( comboBox != null )
{
    comboBox.BeginUpdate();
    comboBox.LabelHeight = axXMLGrid1.NodeHeight - 3;
    comboBox.LinesAtRoot = EXCOMBOBOXLib.LinesAtRootEnum.exLinesAtRoot ;
    comboBox.set_HeightList( null, 256 );
    comboBox.set_WidthList( null, 256 );
    comboBox.IntegralHeight = true;
    comboBox.Columns.Add("Name");
    comboBox.Columns.Add("Value");
    comboBox.ColumnAutoResize = true;
    EXCOMBOBOXLib.Items items = comboBox.Items;
    int h = items.AddItem("Item 1");
    items.set_CellCaption(h, 1, "Item 1.2" );
    int h1 = items.InsertItem(h, null, "SubItem 1");
    items.set_CellCaption(h1, 1,"SubItem 1.2");
    h1 = items.InsertItem(h, null, "SubItem 2");
    items.set_CellCaption(h1, 1,"SubItem 2.2");
    items.set_ExpandItem(h, true);
    comboBox.EndUpdate();
}
```

In C# your project needs a new reference to the Exontrol's ExComboBox control library. Use the Project\Add Reference\COM item to add new reference to a COM object. Once that you added a reference to the Exontrol's ExComboBox the EXCOMBOBOXLib namespace is created. The EXCOMBOBOXLib namespace contains definitions for all objects that ExComboBox control exports.

The following VFP sample closes the Exontrol's ExComboBox user editor when the user selects a new value, or when it presses the Escape key. Also the sample changes the value of the node in the ExComboBox control:

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, node

If (ev.Name = "Change") Then
   node.Value = object.Select(0)
   closeeditor = .t.
else
   If (ev.Name = "KeyPress") Then
      local l
      l = Ev(0).Value
      If l = 27 Then
         closeeditor = .t.
      EndIf
   EndIf
EndIf
```

the sample requires an ActiveX editor (in our case the Exontrol's ExComboBox control):

```
With thisform.XMLGrid1.Editors
   With .Add("excombobox", 16) && UserEditorType
      .UserEditor("Exontrol.ComboBox", "")
      With .UserEditorObject
         .BeginUpdate
         .LabelHeight = thisform.XMLGrid1.NodeHeight - 3
         .LinesAtRoot = -1
         .HeightList(0) = 256
         .WidthList(0) =256
         .IntegralHeight = .t.
         .Columns.Add ("Name")
```

```
        .Columns.Add ("Value")
        .ColumnAutoResize = .t.
        With .Items
           .DefaultItem = .AddItem("Item 1")
           h = .DefaultItem
           .CellCaption(0, 1) = "Item 1.2"
           .DefaultItem = .InsertItem(h, , "SubItem 1")
           .CellCaption(0, 1) = "SubItem 1.2"
           .DefaultItem = .InsertItem(h, , "SubItem 2")
           .CellCaption(0, 1) = "SubItem 2.2"
           .DefaultItem = h
           .ExpandItem(0) = .t.
        EndWith
        .EndUpdate
     EndWith
  EndWith
EndWith
```

event UserEditorOpen (Object as Object, Node as Node)

Occurs when an user editor is about to be opened.

Type	Description
Object as Object	An object created by UserEditor property.
Node as Node	A Node object that hosts an user editor.

The control fires the UserEditorOpen event when an user editor is shown. Use the UserEditorOpen event to initialize the user editor when it is shown. The control fires the UserEditorOleEvent event each time when an user editor fires an event. Use the Add method and UserEditorType type to add an ActiveX editor to the control. Use the UserEditor method to create an ActiveX editor. Use the UserEditorObject property to get the ActiveX editor created by the UserEditor method. The control fires the UserEditorClose event when the user closes the ActiveX editor (for instance, when he clicks outside the editing node).

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

C#
```
private void UserEditorOpen(object sender,object
Obj,exontrol.EXMLGRIDLib.Node Node)
{
}
```

VB
```
Private Sub UserEditorOpen(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Node As exontrol.EXMLGRIDLib.Node) Handles UserEditorOpen
End Sub
```

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

C#
```
private void UserEditorOpen(object sender,
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOpenEvent e)
{
}
```

C++
```
void OnUserEditorOpen(LPDISPATCH Object,LPDISPATCH Node)
{
}
```

C++ Builder
```
void __fastcall UserEditorOpen(TObject *Sender,IDispatch
*Object,Exmlgridlib_tlb::INode *Node)
```

```
{
}
```

Delphi

```
procedure UserEditorOpen(ASender: TObject; Object : IDispatch;Node : INode);
begin
end;
```

Delphi 8 (.NET only)

```
procedure UserEditorOpen(sender: System.Object; e:
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOpenEvent);
begin
end;
```

Powe…

```
begin event UserEditorOpen(oleobject Object,oleobject Node)
end event UserEditorOpen
```

VB.NET

```
Private Sub UserEditorOpen(ByVal sender As System.Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOpenEvent) Handles UserEditorOpen
End Sub
```

VB6

```
Private Sub UserEditorOpen(ByVal Object As Object,ByVal Node As
EXMLGRIDLibCtl.INode)
End Sub
```

VBA

```
Private Sub UserEditorOpen(ByVal Object As Object,ByVal Node As Object)
End Sub
```

VFP

```
LPARAMETERS Object,Node

```

Xbas…

```
PROCEDURE OnUserEditorOpen(oXMLGrid,Object,Node)
RETURN
```

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

Java…

```
<SCRIPT EVENT="UserEditorOpen(Object,Node)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

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

```
Function UserEditorOpen(Object,Node)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComUserEditorOpen Variant llObject Variant llNode
    Forward Send OnComUserEditorOpen llObject llNode
End_Procedure
```

Visual Objects

```
METHOD OCX_UserEditorOpen(Object,Node) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_UserEditorOpen(COM _Object,COM _Node)
{
}
```

XBasic

```
function UserEditorOpen as v (Object as P,Node as
OLE::Exontrol.XMLGrid.1::INode)
end function
```

dBASE

```
function nativeObject_UserEditorOpen(Object,Node)
return
```

The following VB sample selects an item into an user editor of EXCOMBOBOXLibCtl.ComboBox (Exontrol's ExComboBox control) type:

```
Private Sub XMLGrid1_UserEditorOpen(ByVal Object As Object, ByVal Node As
EXMLGRIDLibCtl.INode)
On Error Resume Next
   With Object.Items
      .SelectItem(.FindItem(Node.Value)) = True
   End With
End Sub
```

The following C++ sample selects an item into an user editor of EXCOMBOBOXLibCtl.ComboBox (Exontrol's ExComboBox control) type:

```
#import <excombobox.dll>
void OnUserEditorOpenXmlgrid1(LPDISPATCH Object, LPDISPATCH Node)
```

```
{
   COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
   CNode node( Node ); node.m_bAutoRelease = FALSE;
   EXCOMBOBOXLib::IComboBoxPtr spComboBox = Object;
   if ( spComboBox != NULL )
   {
      long nItem = NULL;
      EXCOMBOBOXLib::IItemsPtr spItems = spComboBox->Items;
      if ( SUCCEEDED( spItems->get_FindItem( node.GetValue(), COleVariant( long(0) ),
vtMissing, &nItem ) ) )
         spItems->put_SelectItem( nItem, VARIANT_TRUE );
   }
}
```

The sample assumes that the Object parameter holds an ExComboBox control. We need to call the #import <excombobox.dll> in order to include definitions for objects and types in the ExComboBox control. The #import <excombobox.dll> creates EXCOMBOBOXLib namespace that includes all definitions for objects and types that the ExComboBox control exports.

The following VB.NET sample selects an item into an user editor of EXCOMBOBOXLibCtl.ComboBox (Exontrol's ExComboBox control) type:

```
Private Sub AxXMLGrid1_UserEditorOpen(ByVal sender As Object, ByVal e As
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOpenEvent) Handles
AxXMLGrid1.UserEditorOpen
   On Error Resume Next
   With e.object.Items
      .SelectItem(.FindItem(e.node.Value)) = True
   End With
End Sub
```

The following C# sample selects an item into an user editor of EXCOMBOBOXLibCtl.ComboBox (Exontrol's ExComboBox control) type:

```
private void axXMLGrid1_UserEditorOpen(object sender,
AxEXMLGRIDLib._IXMLGridEvents_UserEditorOpenEvent e)
{
   EXCOMBOBOXLib.ComboBox comboBox = e.@object as EXCOMBOBOXLib.ComboBox;
```

```
    if (comboBox != null)
    {
        EXCOMBOBOXLib.Items items = comboBox.Items;
        int nItem = items.get_FindItem(e.node.Value, 0, null);
        if (nItem != 0)
            items.set_SelectItem(nItem, true);
    }
}
```

In C# your project needs a new reference to the Exontrol's ExComboBox control library. Use the Project\Add Reference\COM item to add new reference to a COM object. Once that you added a reference to the Exontrol's ExComboBox the EXCOMBOBOXLib namespace is created. The EXCOMBOBOXLib namespace contains definitions for all objects that ExComboBox control exports.

The following VFP sample selects an item into an user editor of EXCOMBOBOXLibCtl.ComboBox (Exontrol's ExComboBox control) type:

```
*** ActiveX Control Event ***
LPARAMETERS object, node

With object.Items
    .DefaultItem = .FindItem(node.Value,0)
    if ( .DefaultItem # 0 )
        .SelectItem(0) = .t.
    endif
EndWith
```