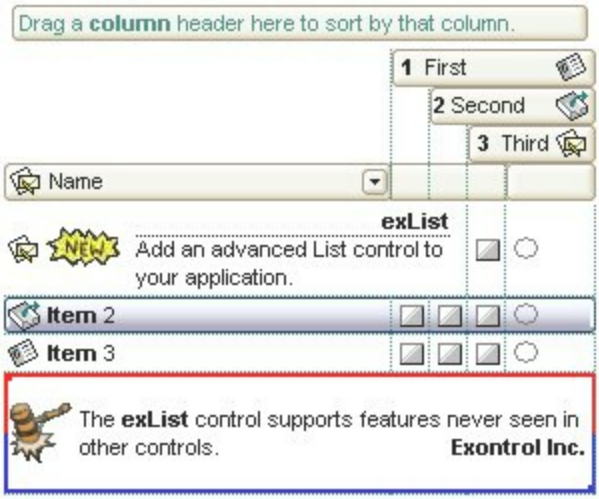
ExList

Add an advanced list control to your application. The exList component displays and edit your tabular data. The exList component lets the user changes its visual appearance using **skins**, each one providing an additional visual experience that enhances viewing pleasure. Skins are relatively easy to build and put on any part of the control.

Features include:

- **Print** and Print Preview support.
- Ability to display more than 2,000,000,000 records, using the virtual mode.
- Any cell supports **Built-in HTML format.**
- **WYSWYG Template/Layout Editor** support.
- **Skinnable Interface** support (ability to apply a skin to the any background part)
- **Filter** support
  - **Filter-Prompt** support, allows you to filter the items as you type while the filter bar is always visible on the bottom part of the list area.
  - **Filter-On-Type** support**.** Ability to filter items by a column, as you type.
  - Ability to filter items using patterns that include **wild card characters** like *, ? or #, **items between two dates, numbers**, **checkboxes** with an easy UI interface.
  - Ability to filter items using **OR, AND** or **NOT operators** between columns.
- **Conditional Format** support.
- **Computed Fields** support.
- 'starts with' and 'contains' **incremental searching** support.
- Ability to assign multiple icons to a cell.
- Ability to load icons and pictures using the BASE64 encoded strings.
- Background Picture support.
- Transparent Selection support.
- Ability to specify how selected items are displayed.
- Ability to show the control's element from right-to-left for Hebrew, Arabic and other **RTL** languages
- support for drag and drop items
- multiple columns
- multiple selection
- multi-line list items
- multi-line HTML tooltip support, XP shadow effect
- **multiple levels header** support
- multiple lines header support
- **sorting by single or multiple columns** support
- user resizable columns
- columns dragable

- locked or unlocked columns
- "**Merge Cells**" support
- unlimited color options for cells or items
- ability to set a font for any cell or item
- cells can be formatted individually, via columns or rows
- radio buttons, images, check boxes
- hyperlink cells support.
- mouse wheel support
- column alignment right, left, or center
- format columns
- break items, items with different heights
- and more



How to get support?

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

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

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

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

Regards,
Exontrol Development Team

https://www.exontrol.com

How to start?

The following steps show you progressively how to start programming the Exontrol's ExList component:

1. **Adding columns**. The control supports multiple columns, so at least one column must be added, before anything else. The Columns.Add method adds a new column to the control's columns collection. Another option to add columns is using the DataSource method of the control. If you have an ADO or DAO recordset just pass it to the DataSource property, and it will do the rest. The AddColumn event notifies your application that a new column has been added. Check the Column object for all options you can apply to a column.
2. **Adding items/data**. The Items object holds a collection of items. Each item is identified by its index. Each item contains a set of cells, one for each column in the control. Each cell is identified by its index in the item. So, an item is always referred as ItemProperty(i), and the cell as CellProperty(i,c). The control provides several ways to add items. If you are using the DataSource method as described in the step 1, the fields from the recordset are automatically loaded to the control. When you are doing manually, use the Items.Add to add new items. The PutItems method takes an array of data and loads it in the control. The AddItem event notifies your application that a new item is added.
3. **Filling the cells.** If your control contains a single column, the data in the column is automatically put at adding time, because any of the AddItem or InsertItem method contains a Caption parameter that may be used at loading time. If you have a control with multiple columns, you need to use the Caption property to specify the captions for the rest of the columns. The Add method may use array of data as parameters in order to specify captions for all cells in the data.
4. **Adding options for cells and items.** The Items object holds the entire collection of options that may be applied to any cell or item. For instance, the CellBold property bolds a cell, the ItemForeColor property changes the foreground color for the entire item, the CellImage property assigns an icon to a cell, or the CellHasCheckBox property assigns a checkbox to a cell.
5. **Adding events**. The control supports events for most of the UI operations. For instance, the user clicks a checkbox, the CellStateChanged event is fired, or the user changes the cell's value so the Change event is fired.

No matter what programming language you are using, you can have a quick view of the component's features using the **WYSWYG Template** editor. It's a nice feature and we don't want you to miss it.

- 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 Template feature lets you to use a simple x-script language to call properties and methods of the control at design as well at runtime. You can use this feature to build x-script strings to pass them at runtime. You can find a short description of the x-script language here

Send comments on this topic.

constants AlignmentEnum

Defines the column's alignment. Use the Alignment property to align the cells in the column.

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

Defines the control's appearance. Use the Appearance property to specify the control's appearance.

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

constants AutoDragEnum

The AutoDragEnum type indicates what the control does when the user clicks and start dragging a row or an item. The AutoDrag property indicates the way the component supports the AutoDrag feature. The AutoDrag feature indicates what the control does when the user clicks an item and start dragging. For instance, using the AutoDrag feature you can automatically lets the user to drag and drop the data to OLE compliant applications like Microsoft Word, Excel and so on. The SingleSel property specifies whether the control supports single or multiple selection. The drag and drop operation starts once the user clicks and moves the cursor up or down, if the SingleSel property is True, and if SingleSel property is False, the drag and drop starts once the user clicks, and waits for a short period of time. If SingleSel property is False, moving up or down the cursor selects the items by drag and drop.

The AutoDragEnum type supports the following values:

Name	Value	Description
exAutoDragNone	0	AutoDrag is disabled. You can use the OLEDropMode property to handle the OLE Drag and Drop event for your custom action.
exAutoDragPosition	1	The item can be dragged from a position to another, but not outside of its group. If your items are arranged as a flat list, no hierarchy, this option can be used to allow the user change the item's position at runtime by drag and drop. This option does not change the parent of any dragged item. The dragging items could be the focused item or a contiguously selection. Click the selection and moves the cursor up or down, so the position of the dragging items is changed. The draggable collection is a collection of sortable items between 2 non-sortable items (SortableItem property). The drag and drop operation can not start on a non-sortable or non-selectable item (SelectableItem property). In other words, you can specify a range where an item can be dragged using the SortableItem property. Just set the SortableItem property on False, for margins, and so the items can be dragged between these items only.
		Drag and drop the selected items to a target application, and paste them as image or text. Pasting the data to the target application depends

exAutoDragCopy	8	on the application. You can use the exAutoDragCopyText to specify that you want to paste as Text, or exAutoDragCopyImage as an image.
exAutoDragCopyText	9	Drag and drop the selected items to a target application, and paste them as text only. Ability to drag and drop the data as text, to your favorite Office applications, like Word, Excel, or any other OLE-Automation compliant. The drag and drop operation can start anywhere Click here to watch a movie on how exAutoDragCopyText works.
exAutoDragCopyImage	10	Drag and drop the selected items to a target application, and paste them as image only. Ability to drag and drop the data as it looks, to your favorite Office applications, like Word, Excel, or any other OLE-Automation compliant. The drag and drop operation can start anywhere Click here to watch a movie on how exAutoDragCopyImage works.
exAutoDragCopySnapShot	11	Drag and drop a snap shot of the current component. This option could be used to drag and drop the current snap shot of the control to your favorite Office applications, like Word, Excel, or any other OLE-Automation compliant.
exAutoDragScroll	16	The component is scrolled by clicking the item and dragging to a new position. This option can be used to allow user scroll the control's content with NO usage of the scroll bar, like on your IPhone. Ability to smoothly scroll the control's content. The feature is useful for touch screens or tables pc, so no need to click the scroll bar in order to scroll the control's content. Use the ScrollBySingleLine property on False, to allow scrolling pixel by pixel when user clicks the up or down buttons on the vertical scroll bar. Click here to watch a movie on how

		exAutoDragScroll works.
exAutoDragPositionOnShortTouch	256	exAutoDragPositionOnShortTouch. The object can be dragged from a position to another, but not outside of its group.
exAutoDragCopyOnShortTouch	2048	exAutoDragCopyOnShortTouch. Drag and drop the selected objects to a target application, and paste them as image or text.
exAutoDragCopyTextOnShortTouch	2304	exAutoDragCopyTextOnShortTouch. Drag and drop the selected objects to a target application, and paste them as text only.
exAutoDragCopyImageOnShortTouch	2560	exAutoDragCopyImageOnShortTouch. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnShortTouch	2816	exAutoDragCopySnapShotOnShortTouch. Drag and drop a snap shot of the current component.
exAutoDragScrollOnShortTouch	4096	exAutoDragScrollOnShortTouch. The component is scrolled by clicking the object and dragging to a new position.
exAutoDragPositionOnRight	65536	exAutoDragPositionOnRight. The object can be dragged from a position to another, but not outside of its group.
exAutoDragCopyOnRight	524288	exAutoDragCopyOnRight. Drag and drop the selected objects to a target application, and paste them as image or text.
exAutoDragCopyTextOnRight	589824	exAutoDragCopyTextOnRight. Drag and drop the selected objects to a target application, and paste them as text only.
exAutoDragCopyImageOnRight	655360	exAutoDragCopyImageOnRight. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnRight	720896	exAutoDragCopySnapShotOnRight. Drag and drop a snap shot of the current component.
exAutoDragScrollOnRight	1048576	exAutoDragScrollOnRight. The component is scrolled by clicking the object and dragging to a new position.
exAutoDragPositionOnLongTouch	16777216	exAutoDragPositionOnLongTouch. The object can be dragged from a position to another, but not

	outside of its group.
exAutoDragCopyOnLongTouch134217728	exAutoDragCopyOnLongTouch. Drag and drop the selected objects to a target application, and paste them as image or text.
exAutoDragCopyTextOnLongTouch150994944	exAutoDragCopyTextOnLongTouch. Drag and drop selected objects to a target application, and paste them as text only.
exAutoDragCopyImageOnLongTouch167772160	exAutoDragCopyImageOnLongTouch. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnLongTouch184549376	exAutoDragCopySnapShotOnLongTouch. Drag and drop a snap shot of the current component.
exAutoDragScrollOnLongTouch268435456	exAutoDragScrollOnLongTouch. The component is scrolled by clicking the object and dragging to a new position.

constants AutoSearchEnum

Specifies the kind of searching while user types characters within a column. Use the AutoSearch property to allow 'start with' incremental search or 'contains' incremental search feature in the control.

Name	Value	Description
exStartWith	0	Defines the 'starts with' incremental search within the column. If the user type characters within the column the control looks for items that start with the typed characters.
exContains	1	Defines the 'contains' incremental search within the column. If the user type characters within the column the control looks for items that contain the typed characters.

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
exHeaderFilterBarButton	0	Specifies the background color for the drop down filter bar button.
exFooterFilterBarButton	1	Specifies the background color for the closing button in the filter bar.
exCellButtonUp	2	Specifies the background color for the cell's button, when it is up.
exCellButtonDown	3	Specifies the background color for the cell's button, 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.
exSelBackColorFilter	20	Specifies the visual appearance for the selection in the drop down filter window. The drop down filter window shows up when the user clicks the filter button in the column's header. Use the DisplayFilterButton property to specify whether the drop down filter bar button is visible or hidden.
exSelForeColorFilter	21	Specifies the foreground color for the selection in the drop down filter window.
exBackColorFilter	26	Specifies the background color for the drop down filter window. If not specified, the BackColorHeader property specifies the drop down filter's background color. Use the exSelBackColorFilter option to specify the selection background visual appearance in the drop down filter window.
exForeColorFilter	27	Specifies the foreground color for the drop down filter window. If not specified, the ForeColorHeader property specifies the drop down filter's foreground color. Use the exSelForeColorFilter option to specify the selection foreground color in the drop down filter window.
exSortBarLinkColor	28	Indicates the color or the visual appearance of the links between columns in the control's sort bar.
exCursorHoverColumn	32	Specifies the visual appearance for the column when the cursor hovers the column.
exDragDropBefore	33	Specifies the visual appearance for the drag and drop cursor before showing the items. This option can be used to apply a background to the dragging items, before painting the items.

exDragDropAfter	34	Specifies the visual appearance for the drag and drop cursor after showing the items. This option can be used to apply a semi-transparent/opaque background to the dragging items, after painting the items. If the exDragDropAfter option is set on white (0x00FFFFFF), the image is not showing on OLE Drag and drop.
exDragDropListTop	35	Specifies the graphic feedback of the item from the drag and drop cursor if the cursor is in the top half of the row. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
exDragDropListBottom	36	Specifies the graphic feedback of the item from the drag and drop cursor if the cursor is in the bottom half of the row. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
exDragDropForeColor	37	Specifies the foreground color for the items being dragged. By default, the foreground color is black.
		Specifies the graphic feedback of the item from the

exDragDropListOver	38	cursor if it is over the item. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
exDragDropListBetween	39	Specifies the graphic feedback of the item when the drag and drop cursor is between items. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
exDragDropAlign	40	Specifies the alignment of the drag and drop image relative to the cursor. By default, the exDragDropAlign option is 0, which initially the drag and drop image is shown centered relative to the position of the cursor. The valid values are listed as follows (hexa representation): • 0x00000000, (default), the drag and drop image is shown centered relative to the cursor, and shows up. • 0x01000000, (left), the drag and drop image is

		shown to the left of the cursor. • 0x02000000, (right), the drag and drop image is shown to the right of the cursor. • 0x04000000, (center-down), the drag and drop image is shown centered relative to the cursor, and shows down. • 0xFF000000, (as- is), the drag and drop image is shown as it is clicked.
exHeaderFilterBarActive	41	exHeaderFilterBarActive. Specifies the visual appearance of the drop down filter bar button, while filter is applied to the column.
exToolTipAppearance	64	Indicates the visual appearance of the borders of the tooltips. 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 CellToolTip property to specify the cell's tooltip. 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.
exListOLEDropPosition	96	By default, the exListOLEDropPosition is 0, which means no effect. Specifies the visual appearance of the dropping position inside the control, when the control is implied in a OLE Drag and Drop operation. The exListOLEDropPosition has effect only if different than 0, and the OLEDropMode property is not exOLEDropNone. For instance, set the Background(exScheduleOLEDropPosition) property on RGB(0,0,255), and a blue line is shown at the item position when the cursor is hover the control, during an OLE Drag and Drop position. The OLEDragDrop event notifies your application once an object is drop in the control.
exSelBackColorHide	166	Specifies the selection's background color, when the control has no focus.
		Specifies the selection's foreground color, when the

exSelForeColorHide	167	control has no focus.
exTreeGlyphOpen	180	Specifies the visual appearance for the +/- buttons when it is collapsed.
exTreeGlyphClose	181	Specifies the visual appearance for the +/- buttons when it is expanded.
exColumnsPositionSign	182	exColumnsPositionSign. Specifies the visual appearance for the position sign between columns, when the user changes the position of the column by drag an drop.
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.
exHSUpperP	401	The upper part (exUpperBackPart) when it is

		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.
exScrollSizeGrip	511	Specifies the visual appearance of the control's size grip when both scrollbars are shown.

constants BackModeEnum

Specifies the background mode when painting the selected items. The SelBackMode property retrieves or sets a value that indicates whether the selection is transparent or opaque.

Name	Value	Description
exOpaque	0	The selection is opaque.
exTransparent	1	The selection is transparent.
exGrid	2	The selection is half transparent half opaque
exCustom	3	User is responsible for painting the selected items.

constants BreakLineEnum

Defines the type of break lines. In order to display an item of break type that caption of the item needs to be empty. Use the ItemBreak property to specify whether an item is a break item.

Name	Value	Description
EmptyLine	0	EmptyLine
SingleLine	1	SingleLine
DoubleLine	2	DoubleLine
DotLine	3	DotLine
DoubleDotLine	4	DoubleDotLine
ThinLine	5	ThinLine
DoubleThinLine	6	DoubleThinLine

constants CaptionFormatEnum

Defines how the cell's caption is painted. Use the CaptionFormat property to specify whether the Caption property supports built-in HTML format.

Name	Value	Description
exText	0	No HTML tags are painted
		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. The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using <a ;exp=> or <a ;e64=> anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption. ◦ exp, stores the plain text to be shown once the user clicks the anchor, such as " <a ;exp=show lines>" ◦ e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAA

" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAA
string encodes the "<fgcolor 808080>show lines<a>-</fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

- ** ... ** 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> ...

exHTML	1	</solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values. • **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values. • **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>). • **<r>** right aligns the text • **<c>** centers the text • ** ** forces a line-break • **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels. • **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.

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

  gradient-center
- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width

		indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<**out** 000000><fgcolor=FFFFFF>outlined</fgcolor></**out**>" generates the following picture:  • **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<**sha**>shadow</**sha**>" generates the following picture:  or "*<**sha** 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></**sha**>*" gets: 
exComputedField	2	Indicates a computed field. The Caption or the ComputedField property indicates the formula to compute the field.

constants CellSingleLineEnum

The CellSingleLineEnum type defines whether the cell's caption is displayed on a single or multiple lines. The CellSingleLine property retrieves or sets a value indicating whether the cell is displayed using one line, or more than one line. The Def(exCellSingleLine) property specifies that all cells in the column display their content using multiple lines. The CellSingleLineEnum type supports the following values:

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

constants CheckStateEnum

Specifies the states for a checkbox in the control.

Name	Value	Description
Unchecked	0	Specifies whether the cell is unchecked.
Checked	1	Specifies whether the cell is checked.
PartialChecked	2	Specifies whether the cell is partial-checked..

constants DefColumnEnum

The Def property retrieves or sets a value that indicates the default value of given properties for all cells in the same column.

Name	Value	Description
exCellHasCheckBox	0	Assigns check boxes to all cells in the column, if it is True. Similar with the CellHasCheckBox property. *(boolean expression, False)*
exCellHasRadioButton	1	Assigns radio buttons to all cells in the column, if it is True. Similar with the CellHasRadioButton property. *(boolean expression, False)*
exCellHasButton	2	Specifies that all cells in the column are buttons, if it is True. Similar with the CellHasButton property. *(boolean expression, False)*
exCellBackColor	4	Specifies the background color for all cells in the column. Use the CellBackColor property to assign a background color for a specific cell. The property has effect only if the property is different than zero. *(long expression)*
exCellForeColor	5	Specifies the foreground color for all cells in the column. Use the CellForeColor property to assign a foreground color for a specific cell. The property has effect only if the property is different than zero. *(long expression)*
exCellVAlignment	6	Specifies the column's vertical alignment. By default, the Def(exCellVAlignment) property is MiddleAlignment. Use the CellVAlignment property to specify the vertical alignment for a particular cell. *(VAlignmentEnum expression, by default*

		MiddleAlignment)
exHeaderBackColor	7	Specifies the column's header background color. The property has effect only if the property is different than zero. Use this option to change the background color for a column in the header area. The exHeaderBackColor option supports skinning, so 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. *(Color expression)*
exHeaderForeColor	8	Specifies the column's header background color. The property has effect only if the property is different than zero. *(Color expression)*
exCellSingleLine	16	Specifies that all cells in the column displays its content into single or multiple lines. Similar with the CellSingleLine property. If using the CellSingleLine / Def(exCellSingleLine) property, we recommend to set the ScrollBySingleLine property on True so all items can be scrolled. *(CellSingleLineEnum type, previously Boolean expression)*
exCaptionFormat	17	Specifies that all cells in the column display text, HTML format, or a computed field. Similar with the CaptionFormat property, (CaptionFormatEnum expression, exText). The ComputedField property indicates the formula to compute the column, when the Def(exCaptionFormat) is exComputedField. *(CaptionFormatEnum expression)*
		Specifies the order of the drawing parts for the entire column. By default, this option is "check,icon,icons,picture,caption", which means that

exCellDrawPartsOrder	34	the cell displays its parts in the following order: check box/ radio buttons (CellHasCheckBox/CellRadioButton), single icon (CellImage), multiple icons (CellImages), custom size picture (CellPicture), and the cell's caption. Use the exCellDrawPartsOrder option to specify a new order for the drawing parts in the cells of the column. The RightToLeft property automatically flips the order of the columns. (string expression, "check,icon,icons,picture,caption") *(String expression)*
exCellPaddingLeft	48	The padding defines the space between the element border and the element content. Gets or sets the left padding (space) of the cells within the column. This option applies a padding to all cells in the column. Use the exHeaderPaddingLeft option to apply the padding to the column's caption in the control's header. The padding does not affect the element's background color. By default, the exCellPaddingLeft property is 0. *(Long expression)*
exCellPaddingRight	49	Gets or sets the right padding (space) of the cells within the column. This option applies a padding to all cells in the column. Use the exHeaderPaddingRight option to apply the padding to the column's caption in the control's header. The padding does not affect the element's background color. By default, the exCellPaddingRight property is 0. *(Long expression)*
exCellPaddingTop	50	Gets or sets the top padding (space) of the cells within the column. This option applies a padding to all cells in the column. Use the exHeaderPaddingTop option to apply the padding to the column's caption in the control's header. The padding does not affect the element's background color. By default, the exCellPaddingTop property is 0.

		(Long expression)
exCellPaddingBottom	51	Gets or sets the bottom padding (space) of the cells within the column. This option applies a padding to all cells in the column. Use the exHeaderPaddingBottom option to apply the padding to the column's caption in the control's header. The padding does not affect the element's background color. By default, the exCellPaddingBottom property is 0. *(Long expression)*
exHeaderPaddingLeft	52	Gets or sets the left padding (space) of the column's header. This option applies the padding to the column's caption in the control's header. Use the exCellPaddingLeft option to apply the padding to all cells in the column. The padding does not affect the element's background color. By default, the exHeaderPaddingLeft property is 0. *(Long expression)*
exHeaderPaddingRight	53	Gets or sets the right padding (space) of the column's header. This option applies the padding to the column's caption in the control's header. Use the exCellPaddingRight option to apply the padding to all cells in the column. The padding does not affect the element's background color. By default, the exHeaderPaddingRight property is 0. *(Long expression)*
exHeaderPaddingTop	54	Gets or sets the top padding (space) of the column's header. This option applies the padding to the column's caption in the control's header. Use the exCellPaddingTop option to apply the padding to all cells in the column. The padding does not affect the element's background color. By default, the exHeaderPaddingTop property is 0. *(Long expression)*

exHeaderPaddingBottom	55	Gets or sets the bottom padding (space) of the column's header. This option applies the padding to the column's caption in the control's header. Use the exCellPaddingBottom option to apply the padding to all cells in the column. The padding does not affect the element's background color. By default, the exHeaderPaddingBottom property is 0. *(Long expression)*
exColumnResizeContiguously	64	Gets or sets a value that indicates whether the control's content is updated while the user is resizing the column. *(Boolean expression, False)*

constants DescriptionTypeEnum

The control's Description property defines descriptions for few control parts.

Name	Value	Description
exFilterBarAll	0	Defines the caption of (All) in the filter bar window. If the Description(exFilterBarAll) property is empty, the (All) predefined item is not shown in the drop down filter window.
exFilterBarBlanks	1	Defines the caption of (Blanks) in the filter bar window. If the Description(exFilterBarBlanks) property is empty, the (Blanks) predefined item is not shown in the drop down filter window.
exFilterBarNonBlanks	2	Defines the caption of (NonBlanks) in the filter bar window. If the Description(exFilterBarNonBlanks) property is empty, the (NonBlanks) predefined item is not shown in the drop down filter window.
exFilterBarFilterForCaption	3	Defines the caption of "Filter For:" in the filter bar window.
exFilterBarFilterTitle	4	Defines the title for the filter tooltip.
exFilterBarPatternFilterTitle	5	Defines the title for the filter pattern tooltip.
exFilterBarTooltip	6	Defines the tooltip for filter window.
exFilterBarPatternTooltip	7	Defines the tooltip for filter pattern window
exFilterBarFilterForTooltip	8	Defines the tooltip for "Filter For:" window
exFilterBarIsBlank	9	Defines the caption of the function 'IsBlank' in the control's filter bar.
exFilterBarIsNonBlank	10	Defines the caption of the function 'not IsBlank' in the control's filter bar.
exFilterBarAnd	11	Customizes the 'and' operator in the control's filter bar when multiple columns are used to filter the items in the control.
exFilterBarDate	12	Specifies the "Date:" caption being displayed in the drop down filter window when DisplayFilterPattern property is True, and DisplayFilterDate property is True.
		Specifies the "to" sequence being used to split the from date and to date in the Date field of the drop down filter window. For instance, the "to

exFilterBarDateTo	13	12/13/2004" specifies the items before 12/13/2004, "12/23/2004 to 12/24/2004" filters the items between 12/23/2004 and 12/24/2004, or "Feb 12 2004 to" specifies all items after a date.
exFilterBarDateTooltip	14	Describes the tooltip that shows up when cursor is over the Date field. "You can filter the items into a given interval of dates. For instance, you can filter all items dated before a specified date (**to 2/13/2004**), or all items dated after a date (**Feb 13 2004 to**) or all items that are in a given interval (**2/13/2004 to 2/13/2005**)."
exFilterBarDateTitle	15	Describes the title of the tooltip that shows up when the cursor is over the Date field. By default, the exFilterBarDateTitle is "Date".
exFilterBarDateTodayCaption	16	Specifies the caption for the 'Today' button in a date filter window. By default, the exFilterBarDateTodayCaption property is "Today".
exFilterBarDateMonths	17	Specifies the name for months to be displayed in a date filter window. The list of months should be delimitated by space characters. By default, the exFilterBarDateMonths is "January February March April May June July August September October November December".
exFilterBarDateWeekDays	18	Specifies the shortcut for the weekdays to be displayed in a date filter window. The list of shortcut for the weekdays should be separated by space characters. By default, the exFilterBarDateWeekDays is "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.
exFilterBarChecked	19	Defines the caption of (Checked) in the filter bar window. The exFilterBarChecked option is displayed only if the FilterType property is exCheck. If the Description(exFilterBarChecked) property is empty, the (Checked) predefined item is not shown in the drop down filter window. If the user selects the (Checked) item the control filter checked items. The CellState property indicates the state of the cell's checkbox.

exFilterBarUnchecked	20	Defines the caption of (Unchecked) in the filter bar window. The exFilterBarUnchecked option is displayed only if the FilterType property is exCheck. If the Description(exFilterBarUnchecked) property is empty, the (Unchecked) predefined item is not shown in the drop down filter window. If the user selects the (Unchecked) item the control filter unchecked items. The CellState property indicates the state of the cell's checkbox.
exFilterBarIsChecked	21	Defines the caption of the 'IsChecked' function in the control's filter bar. The 'IsChecked' function may appear only if the user selects (Checked) item in the drop down filter window, when the FilterType property is exCheck
exFilterBarIsUnchecked	22	Defines the caption of the 'not IsChecked' function in the control's filter bar. The 'not IsChecked' function may appear only if the user selects (Unchecked) item in the drop down filter window, when the FilterType property is exCheck
exFilterBarOr	23	Customizes the 'or' operator in the control's filter bar when multiple columns are used to filter the items in the control.
exFilterBarNot	24	Customizes the 'not' operator in the control's filter bar.
exFilterBarExclude	25	Specifies the 'Exclude' caption being displayed in the drop down filter. The Exclude option is displayed in the drop down filter window only if the FilterList property includes the exShowExlcude flag.

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	This one is not implemented.

constants exOLEDropModeEnum

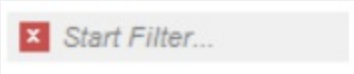
Constants for the OLEDropMode property, that defines how the control accepts OLE drag and drop operations. Use the OLEDropMode property to set how the component handles drop operations.

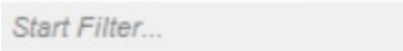
Name	Value	Description
exOLEDropNone	0	The control is not used OLE drag and drop functionality.
exOLEDropManual	1	The control triggers the OLE drop events, allowing the programmer to handle the OLE drop operation in code.

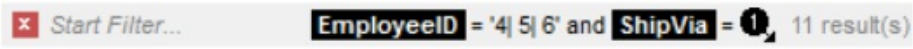
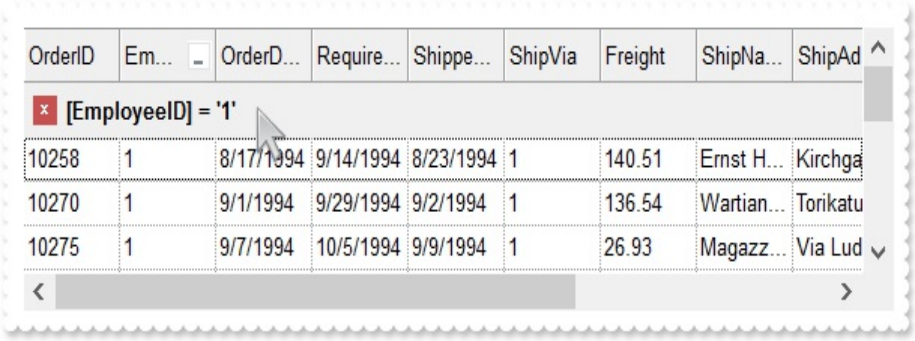
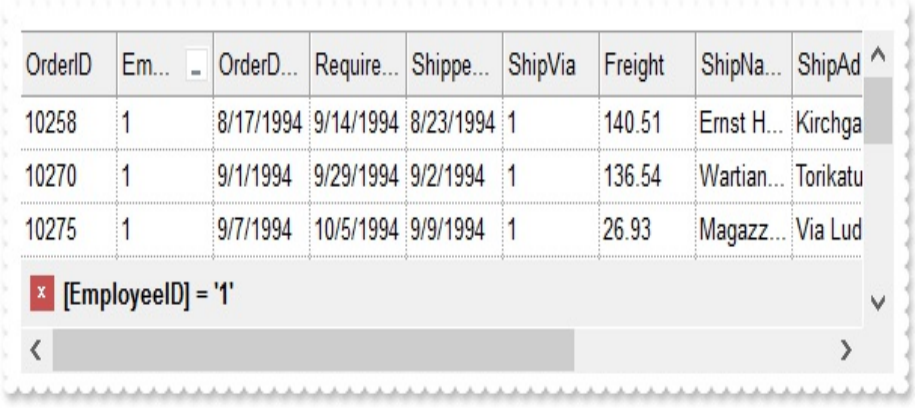
Here's the list of events related to OLE drag and drop: OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag.

constants FilterBarVisibleEnum

The FilterBarVisibleEnum type defines the flags you can use on FilterBarPromptVisible property. The FilterBarCaption property defines the caption to be displayed on the control's filter bar. The FilterBarPromptVisible property , specifies how the control's filter bar is displayed and behave. The FilterBarVisibleEnum type includes several flags that can be combined together, as described bellow:

Name	Value	Description
exFilterBarHidden	0	No filter bar is shown while there is no filter applied. The control's filter bar is automatically displayed as soon a a filter is applied.
exFilterBarPromptVisible	1	The exFilterBarPromptVisible flag specifies that the control's filter bar displays the filter prompt. The exFilterBarPromptVisible, exFilterBarVisible, exFilterBarCaptionVisible flag , forces the control's filter-prompt, filter bar or filter bar description (even empty) to be shown. If missing, no filter prompt is displayed. The FilterBarPrompt property to specify the HTML caption being displayed in the filter bar when the filter pattern is missing. 
exFilterBarVisible	2	The exFilterBarVisible flag forces the control's filter bar to be shown, no matter if any filter is applied. If missing, no filter bar is displayed while the control has no filter applied.  or combined with exFilterBarPromptVisible 
exFilterBarCaptionVisible	4	The exFilterBarVisible flag forces the control's filter bar to display the FilterBarCaption property. 

exFilterBarSingleLine	16	The exFilterBarVisible flag specifies that the caption on the control's filter bar id displayed on a single line. The exFilterBarSingleLine flag , specifies that the filter bar's caption is shown on a single line, so HTML tag or \r\n are not handled. By default, the control's filter description applies word wrapping. Can be combined to exFilterBarCompact to display a single-line filter bar. If missing, the caption on the control's filter bar is displayed on multiple lines. You can change the height of the control's filter bar using the FilterBarHeight property.
exFilterBarToggle	256	The exFilterBarToggle flag specifies that the user can close the control's filter bar (removes the control's filter) by clicking the close button of the filter bar or by pressing the CTRL + F, while the control's filter bar is visible. If no filter bar is displayed, the user can display the control's filter bar by pressing the CTRL + F key. While the control's filter bar is visible the user can navigate though the list or control's filter bar using the ALT + Up/Down keys. If missing, the control's filter bar is always shown if any of the following flags is present exFilterBarPromptVisible, exFilterBarVisible, exFilterBarCaptionVisible.
exFilterBarShowCloseIfRequired	512	The exFilterBarShowCloseIfRequired flag indicates that the close button of the control's filter bar is displayed only if the control has any currently filter applied. The Background(exFooterFilterBarButton) property on -1 hides permanently the close button of the control's filter bar. 
exFilterBarShowCloseOnRight	1024	The exFilterBarShowCloseOnRight flag specifies that the close button of the control's filter bar should be displayed on the right side. If the control's RightToLeft property is True, the close button of the control's filter bar would be automatically displayed on the left side. 

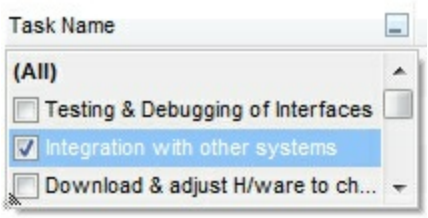
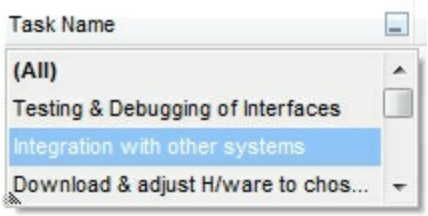
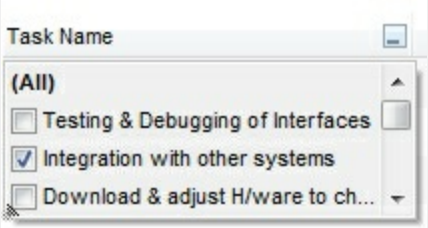
exFilterBarCompact	2048	The exFilterBarCompact flag compacts the control's filter bar, so the filter-prompt will be displayed to the left, while the control's filter bar caption will be displayed to the right. This flag has effect only if combined with the exFilterBarPromptVisible. This flag can be combined with the exFilterBarSingleLine flag, so all filter bar will be displayed compact and on a single line. 
exFilterBarTop	8192	The exFilterBarTop flag displays the filter-bar on top (between control's header and items section as shown:  By default, the filter-bar is shown aligned to the bottom (between items and horizontal-scroll bar) as shown: 

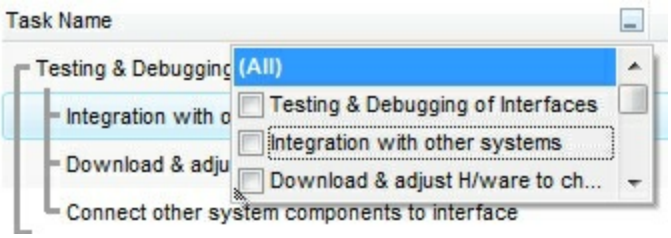
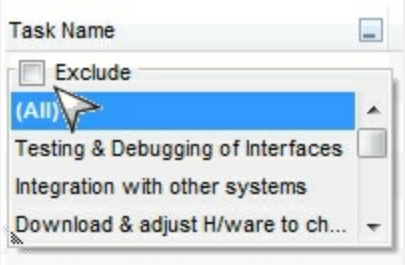
constants FilterListEnum

The FilterListEnum type specifies the type of items being included in the column's drop down list filter. The FilterList property specifies the items being included to the column's drop down filter-list, including other options for filtering. Use the DisplayFilterPattern and/or DisplayFilterDate property to display the pattern field, a date pattern or a calendar control inside the drop down filter window.

The FilterList can be a bit-combination of exAllItems, exVisibleItems or exNoItems with any other flags being described bellow:

Name	Value	Description
exAllItems	0	The filter's list includes all items in the column.
exVisibleItems	1	The filter's list includes only visible (filtered) items from the column. The visible items include child items of collapsed items.
exNoItems	2	The filter's list does not include any item from the column. Use this option if the drop down filter displays a calendar control for instance.
exLeafItems	3	Not implemented.
exRootItems	4	Not implemented.
exSortItemsDesc	16	If the exSortItemsDesc flag is set the values in the drop down filter's list gets listed descending. If none of the exSortItemsAsc or exSortItemsDesc is present, the list is built as the items are displayed in the control.
exSortItemsAsc	32	If the exSortItemsAsc flag is set the values in the drop down filter's list gets listed ascending. If none of the exSortItemsAsc or exSortItemsDesc is present, the list is built as the items are displayed in the control.
exSingleSel	128	If this flag is present, the filter's list supports single selection. By default, (If missing), the user can select multiple items using the CTRL key. Use the exSingleSel property to prevent multiple items selection in the drop down filter list.
		The filter's list displays a check box for each included item. Clicking the checkbox, makes the item to be include din the filter. If this flag is present, the filter is closed once the user presses

exShowCheckBox	256	ENTER or clicks outside of the drop down filter window. By default, (this flag is missing), clicking an item closes the drop down filter, if the CTRL key is not pressed. This flag can be combined with exHideCheckSelect. The following screen shot shows the drop down filter **with** or **with no** exShowCheckBox flag:  or 
exHideCheckSelect	512	The selection background is not shown for checked items in the filter's list. This flag can be combined with exShowCheckBox. The following screen shot shows no selection background for the checked items: 
exShowFocusItem	1024	This flag allows highlighting the focus cell value in the filter's list. The focus cell value is the cell's content at the moment the drop down filter window is shown. For instance, click an item so a new item is selected, and click the drop down filter button. A item being focused in the drop down filter list is the one you have in the control's selection. This flag has effect also, if displaying a calendar control in the drop down filter list. The following screen shot shows the focused item

		in the filter's list (The Integration ... item in the background is the focused item, and the same is in the filter's list) : 
exShowPrevSelectOpaque	2048	By default, the previously selection in the drop down filter's list is shown using a semi-transparent color. Use this flag to show the previously selection using an opaque color. The exSelFilterForeColor and exSelFilterBackColor options defines the filter's list selection foreground and background colors.
exEnableToolTip	4096	This flag indicates whether the filter's tooltip is shown. The Description(exFilterBarTooltip,exFilterBarPatternToolt ...) properties defines the filter's tooltips.
exShowExclude	8192	This flag indicates whether the Exclude option is shown in the drop down filter window. This option has effect also if the drop down filter window shows a calendar control. The following screen shot shows the Exclude field in the drop down filter window: 
exShowBlanks	16384	This flag indicates whether the (Blanks) and (NonBlanks) items are shown in the filter's list

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 FilterBarPromptColumns specifies the list of columns to be used when filtering. 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. Can be combined with exFilterPromptCaseSensitive. 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

exFilterPromptCaseSensitive	256	Filtering the list is case sensitive. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith, exFilterPromptEndWith or exFilterPromptPattern.
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 FilterTypeEnum

Defines the type of filter applies to a column. Use the FilterType property of the Column object to specify the type of filter being used. Use the Filter property of Column object to specify the filter being used. The value for Filter property depends on the FilterType property.

Name	Value	Description
exAll	0	No filter applied
exBlanks	1	Only blank items are included
exNonBlanks	2	Only non blanks items are included
exPattern	3	Only items that match the pattern are included. The Filter property defines the pattern. A pattern may contain the wild card characters '?' for any single character, '*' for zero or more occurrences of any character, '#' for any digit character. If any of the *, ?, # or | characters are preceded by a \ (escape character) it masks the character itself.
exDate	4	Use the exDate type to filter items into a given interval. The Filter property of the Column object defines the interval of dates being used to filter items. The interval of dates should be as [dateFrom] to [dateTo]. Use the Description property to changes the "to" conjunction used to split the dates in the interval. If the dateFrom value is missing, the control includes only the items before the dateTo date, if the dateTo value is missing, the control includes the items after the dateFrom date. If both dates (dateFrom and dateTo) are present, the control includes the items between this interval of dates. For instance, the "2/13/2004 to" includes all items after 2/13/2004 inclusive, or "2/13/2004 to Feb 14 2005" includes all items between 2/13/2004 and 2/14/2004.
exNumeric	5	If the FilterType property is exNumeric, the Filter property may include operators like <, <=, =, <>, >= or > and numbers to define rules to include numbers in the control's list. For instance, the "> 10 < 100" filter indicates all numbers greater than 10 and less than 100.
		Only checked or unchecked items are included. The

exCheck	6	CellState property indicates the state of the cell's checkbox. The control filters for checked items, if the Filter property is "1". The control filters for unchecked items, if the Filter property is "0". A checked item has the the CellState property different than zero. An unchecked item has the CellState property on zero.
exImage	10	Filters items by icons. The CellImage property indicates the cell's icon.
exFilter	240	Only the items that are in the Filter property are included.
exFilterDoCaseSensitive	256	By default, the filtering is case-insensitive. If this flag is set, the filtering is case-sensitive. This option can be combined with exFilter or exPattern flag to perform a case-sensitive filtering. For instance, the exFilter + exFilterDoCaseSensitive indicates that the column includes only the values that match exactly the values in the Filter property.

constants FormatApplyToEnum

The FormatApplyToEnum expression indicates whether a format is applied to an item or to a column. Any value that's greater than 0 indicates that the conditional format is applied to the column with the value as index. A value less than zero indicates that the conditional format object is applied to items. Use the ApplyTo property to specify whether the conditional format is applied to items or to columns.

Name	Value	Description
exFormatToItems	-1	Specifies whether the condition is applied to items.
exFormatToColumns	0	Specifies whether the condition is applied to columns. The 0 value indicates that the conditional format is applied to the first column. The 1 value indicates the conditional format is applied to the second column. The 2 value indicates the conditional format is applied to the third column, and so on.

constants GridLinesEnum

Defines how the control paints the grid lines. Use the DrawGridLines property to specify whether the control draws the grid lines.

Name	Value	Description
exNoLines	0	The control displays no grid lines.
exAllLines	-1	The control displays vertical and horizontal grid lines.
exHLines	1	Only horizontal grid lines are shown.
exVLines	2	Only vertical grid lines are shown.

constants GridLinesStyleEnum

The GridLinesStyle type specifies the style to show the control's grid lines. The GridLineStyle property indicates the style of the gridlines being displayed in the view if the DrawGridLines property is not zero. The GridLinesStyle enumeration specifies the style for horizontal or/and vertical gridlines in the control.

Name	Value	Description
exGridLinesDot	0	The control's gridlines are shown as dotted.
exGridLinesHDot4	1	The horizontal control's gridlines are shown as dotted.
exGridLinesVDot4	2	The vertical control's gridlines are shown as dotted.
exGridLinesDot4	3	The control's gridlines are shown as solid.
exGridLinesHDash	4	The horizontal control's gridlines are shown as dashed.
exGridLinesVDash	8	The vertical control's gridlines are shown as dashed.
exGridLinesDash	12	The control's gridlines are shown as dashed.
exGridLinesHSolid	16	The horizontal control's gridlines are shown as solid.
exGridLinesVSolid	32	The vertical control's gridlines are shown as solid.
exGridLinesSolid	48	The control's gridlines are shown as solid.
exGridLinesGeometric	512	The control's gridlines are drawn using a geometric pen. The exGridLinesGeometric flag can be combined with any other flag. A geometric pen can have any width and can have any of the attributes of a brush, such as dithers and patterns. A cosmetic pen can only be a single pixel wide and must be a solid color, but cosmetic pens are generally faster than geometric pens. The width of a geometric pen is always specified in world units. The width of a cosmetic pen is always 1.

constants HitTestInfoEnum

The HitTestInfoEnum expression defines the hit area within a cell. Use the ItemFromPoint property to determine the hit test code within the cell.

Name	Value	Description
exHTCell	0	In the cell's client area.
exHTCellInside	4	On the icon, picture, check or caption associated with a cell.
exHTCellCaption	20	In the caption associated with a cell.
exHTCellCheck	36	In the check/radio button associated with a cell.
exHTCellIcon	68	In icon associated with a cell.
exHTCellPicture	132	In a picture associated to a cell.
exHTCellCaptionIcon	1044	In the icon's area inside the cell's caption. Use the <img> HTML tag to insert icons inside the cell's caption, when the CaptionFormat property is exHTML.
exHTBottomHalf	2048	(HEXA 800) The cursor is in the bottom half of the row. If this flag is not set, the cursor is in the top half of the row. This is an OR combination with the rest of predefined values. For instance, you can check if the cursor is in the bottom half of the row using HitTestCode AND 0x800
exHTBetween	4096	The cursor is between two rows. This is an OR combination with the rest of predefined values. For instance, you can check if the cursor is between two items using HitTestCode AND 0x1000

constants PictureDisplayEnum

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

Name	Value	Description
UpperLeft	0	Aligns the picture to the upper left corner.
UpperCenter	1	Centers the picture on the upper edge.
UpperRight	2	Aligns the picture to the upper right corner.
MiddleLeft	16	Aligns horizontally the picture on the left side, and centers the picture vertically.
MiddleCenter	17	Puts the picture on the center of the source.
MiddleRight	18	Aligns horizontally the picture on the right side, and centers the picture vertically.
LowerLeft	32	Aligns the picture to the lower left corner.
LowerCenter	33	Centers the picture on the lower edge.
LowerRight	34	Aligns the picture to the lower right corner.
Tile	48	Tiles the picture on the source.
Stretch	49	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 ScrollBars property to specify whether the vertical or horizontal scroll bar is visible or hidden. 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.

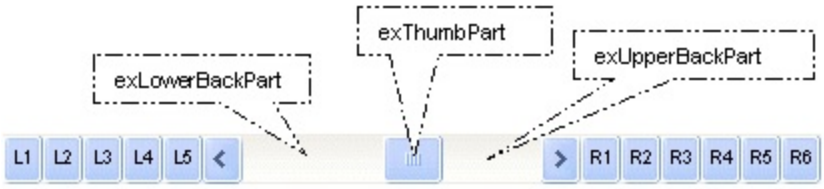
constants ScrollBarsEnum

Specifies the type of scroll bars that control uses. Use the ScrollBars property to specify the control's scroll bars.

Name	Value	Description
NoScroll	0	No scroll bars are shown
Horizontal	1	Only horizontal scroll bars are shown.
Vertical	2	Only vertical scroll bars are shown.
Both	3	Both horizontal and vertical scroll bars are shown.
DisableNoHorizontal	5	The horizontal scroll bar is always shown, it is disabled if it is unnecessary.
DisableNoVertical	10	The vertical scroll bar is always shown, it is disabled if it is unnecessary.
DisableBoth	15	Both horizontal and vertical scroll bars are always shown, disabled if they are unnecessary.

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
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 SortOnClickEnum

Specifies the action that control takes when user clicks the column's header. The SortOnClick Property specifies whether the control sorts a column when its caption has been clicked.

Name	Value	Description
exNoSort	0	The column is not sorted when user clicks the column's header.
exDefaultSort	-1	(default) The control sorts the column when user clicks the column's header.
exUserSort	1	The control displays the sort icons, but it doesn't sort the column.

constants SortOrderEnum

Specifies the column's order type.

Name	Value	Description
SortNone	0	The column is not sorted.
SortAscending	1	The column is sorted ascending.
SortDescending	2	The column is sorted descending.

constants SortTypeEnum

Defines how a column can be sorted. The SortType property returns or sets a value that indicates the way the control sorts the values for a column.

Name	Value	Description
SortString	0	(Default) Values are sorted as strings.
SortNumeric	1	Values are sorted as numbers. Any non-numeric value is evaluated as 0.
SortDate	2	Values are sorted as dates. Group ranges are one day.
SortDateTime	3	Values are sorted as dates and times. Group ranges are one second.
SortTime	4	Values are sorted using the time part of a date and discarding the date. Group ranges are one second.
SortUserData	5	The values sorted are the user data of cells. Values are sorted as numbers.
exSortByValue	16	The column gets sorted by cell's value rather than cell's caption.
exSortByState	32	The column gets sorted by cell's state rather than cell's caption.
exSortByImage	48	The column gets sorted by cell's image rather than cell's caption.

constants ItemsAllowSizingEnum

The ItemsAllowSizingEnum type specifies whether the user can resize items individuals or all items at once, at runtime. Use the ItemsAllowSizing property to specify whether the user can resize items individuals or all items at once, at runtime. Curently, the ItemsAllowSizingEnum type supports the following values:

Name	Value	Description
exNoSizing	0	The user can't resize the items at runtime.
exResizeItem	-1	Specifies whether the user resizes the item from the cursor.
exResizeAllItems	1	Specifies whether the user resizes all items at runtime.

constants UIVisualThemeEnum

The UIVisualThemeEnum expression specifies the UI parts that the control can shown using the current visual theme. The UseVisualThemeproperty 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
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme

constants VAlignmentEnum

Specifies the vertical alignment for the fields (captions). Use the CellVAlignment property to align vertically the cell's caption.

Name	Value	Description
TopAlignment	0	The field is top aligned.
MiddleAlignment	1	The field is centered.
BottomAlignment	2	The field is bottom aligned.

Appearance object

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

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

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

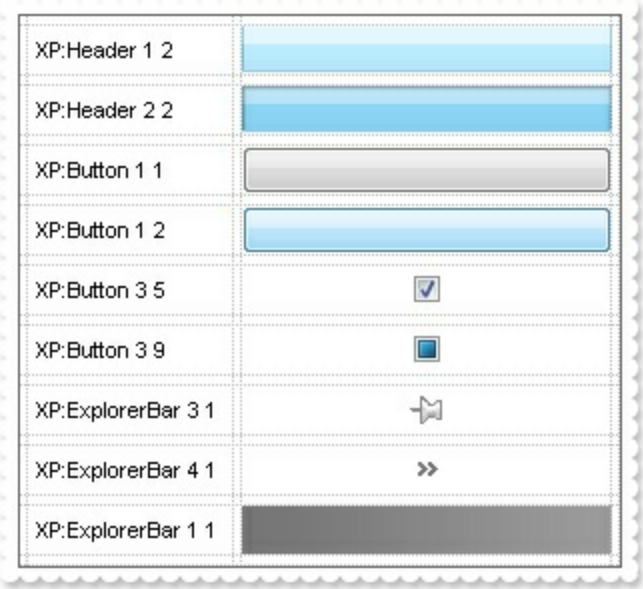
Adds or replaces a skin object to the control.

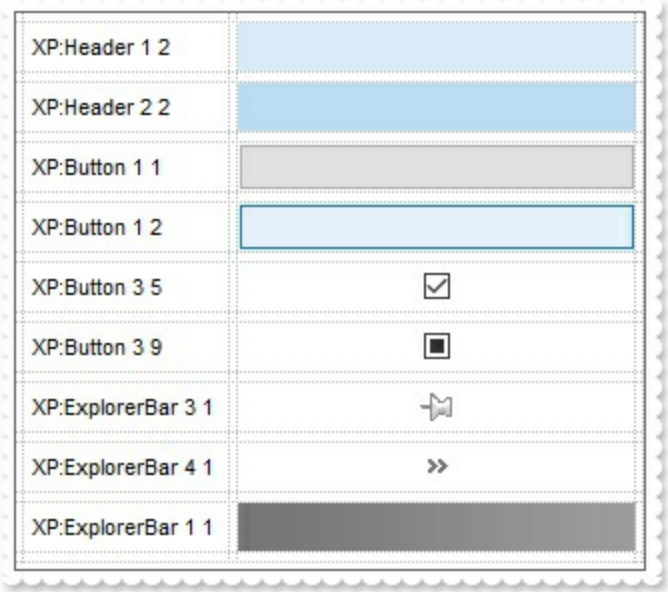
Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
	The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use the BYTE[] / safe arrays of VT_I1 or VT_UI1 option when using the EBN file directly in the resources of the project. For instance, the VB6 provides the LoadResData to get the safe array o bytes for specified resource, while in VB/NET or C# the internal class Resources provides definitions for all files being inserted. (ResourceManager.GetObject("ebn", resourceCulture)) If the Skin parameter points to a string expression, it can be one of the following: • A path to the skin file (*.EBN). The ExButton component or ExEBN tool can be used to create, view or edit EBN files. For instance, "C:\Program Files\Exontrol\ExButton\Sample\EBN\MSOffice-Ribbon\msor_frameh.ebn" • A BASE64 encoded string that holds the skin file (*.EBN). Use the ExImages tool to build BASE 64 encoded strings of the skin file (*.EBN). The BASE64 encoded string starts with "gBFLBCJw..." • An Windows XP theme part, if the Skin parameter starts with "**XP:**". Use this option, to display any UI element of the Current Windows XP Theme, on any part of the control. In this case, the syntax of the Skin parameter is: "XP:ClassName Part State" where the ClassName defines the window/control class name in the Windows XP Theme, the Part indicates a long expression that defines the part, and the State indicates the state of the part to be shown. All known values for window/class, part and start are defined at

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

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

Skin as Variant

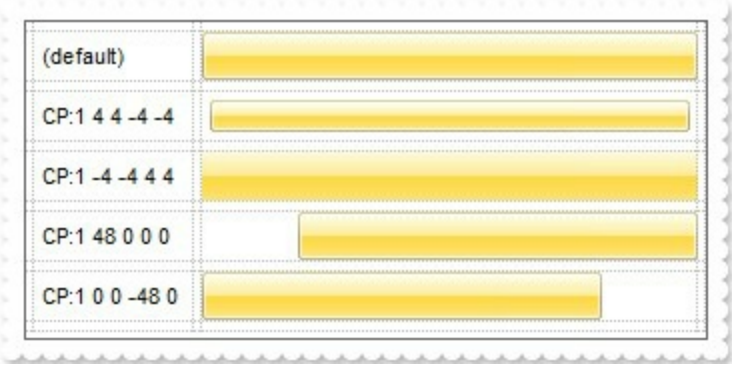




- A copy of another skin with different coordinates (position, size), if the Skin parameter starts with "**CP**:". Use this option, to display the EBN, using different coordinates (position, size). By default, the EBN skin object is rendered on the part's client area. Using this option, you can display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

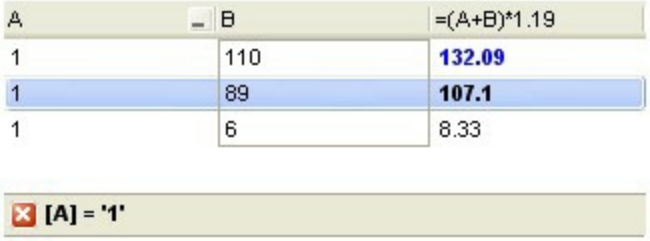
where the ID is the identifier of the EBN to be used (it is a number that specifies the ID parameter of the Add method), Left, Top, Right and Bottom parameters/numbers specifies the relative position to the part's client area, where the EBN should be rendered. The Left, Top, Right and Bottom parameters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

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

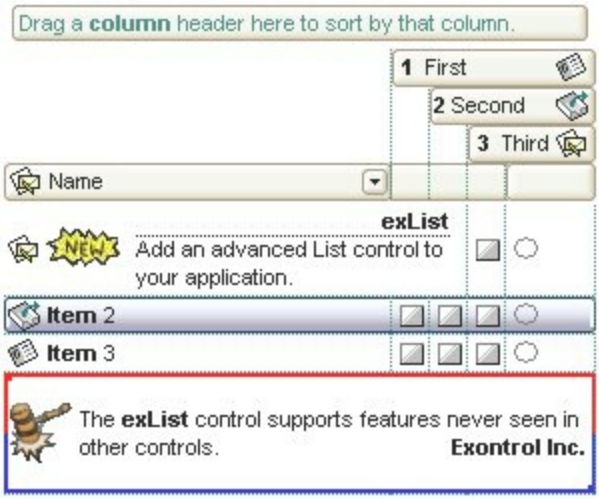


Return	Description
Boolean	A Boolean expression that indicates whether the new skin was added or replaced.

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (*.ebn) assigned to a part of the control, when the "XP:" prefix is not specified in the Skin parameter (available for Windows XP systems). By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part. Use the Remove method to remove a specific skin from the control. Use the Clear method to remove all skins in the control. Use the BeginUpdate and EndUpdate methods to maintain performance while do multiple changes to the control. 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:

- control's border, Appearance property
- control's **header bar**, BackColorHeader property
- control's **filter bar**, FilterBarBackColor property
- control's **sort bar**, BackColorSort property
- the caption of the control's sort bar, BackColorSortCaption property
- **selected item** or cell, SelBackColor property
- **item**, ItemBackColor property
- **cell**, CellBackColor property
- cell's **button**, "**drop down**" filter bar button, "close" filter bar button, tooltips, and so on, Background property

- CellImage, CellImages, HeaderImage, CheckImage or RadioImage property

The following VB sample changes the visual appearance for the selected item. 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 item(s):

```
With List1
   With .VisualAppearance
      .Add &H23, App.Path + "\selected.ebn"
   End With
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = &H23000000
End With
```

The sample adds the skin with the index 35 (Hexa 23), and applies to the selected item using the SelBackColor property.

The following C++ sample applies a new appearance to the selected item(s):

```
#include "Appearance.h"
m_list.GetVisualAppearance().Add( 0x23,
COleVariant(_T("D:\\Temp\\ExList_Help\\selected.ebn")) );
m_list.SetSelBackColor( 0x23000000 );
m_list.SetSelForeColor( 0 );
```

The following VB.NET sample applies a new appearance to the selected item(s):

```
With AxList1
   With .VisualAppearance
      .Add(&H23, "D:\Temp\ExList_Help\selected.ebn")
   End With
   .SelForeColor = Color.Black
   .Template = "SelBackColor = 587202560"
End With
```

The VB.NET sample uses the Template property to assign a new value to the SelBackColor property. The 587202560 value represents &23000000 in hexadecimal.

The following C# sample applies a new appearance to the selected item(s):

```
axList1.VisualAppearance.Add(0x23, "D:\\Temp\\ExList_Help\\selected.ebn");
axList1.Template = "SelBackColor = 587202560";
```

The following VFP sample applies a new appearance to the selected item(s):

```
With thisform.List1
   With .VisualAppearance
      .Add(35, "D:\Temp\ExList_Help\selected.ebn")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = 587202560
EndWith
```

The 587202560 value represents &23000000 in hexadecimal. The 32 value represents &23 in hexadecimal

The first screen shot was generated using the following template (On Windows XP):

```
BeginUpdate

ShowFocusRect = Fase
VisualAppearance.Add(1,"XP:Header 1 1")
VisualAppearance.Add(2,"XP:ScrollBar 2 1")
VisualAppearance.Add(3,"XP:Window 18 1")
VisualAppearance.Add(4,"XP:Window 16 1")
BackColorHeader = 16777216
SelBackColor = 33554432
Background(1) = 50331648
Background(0) = 67108864
Background(20) = 33554432
Background(21) = 1
SelForeColor = 0

ConditionalFormats
{
   Add("%2>100")
   {
      Bold = True
```

```
        ForeColor =RGB(0,0,255)
        ApplyTo = 2
    }
}
Columns
{
    "A".DisplayFilterButton = True
    "B"
    "=(A+B)*1.19"
    {
        ComputedField = "(%0 + %1)*1.19"
    }
}
Items
{
    Dim h
    h = Add(1)
    Caption(h,1) = 110
    h = Add(2)
    Caption(h,1) = 22
    h = Add(2)
    Caption(h,1) = 99
    h = Add(1)
    Caption(h,1) = 89
    h = Add(3)
    Caption(h,1) = 11
    h = Add(1)
    Caption(h,1) = 6
}
EndUpdate
```

The second screen shot was generated using the following template:

```
BeginUpdate

Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl
```

```
Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl

Images("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlkt


VisualAppearance
{
   ' Header
   Add(1,
"gBFLBCJwBAEHhEJAEGg4BcoDg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK

   ' HeaderFilterBarButton

Add(2,"gBFLBCJwBAEHhEJAEGg4BCwEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA


Add(3,"gBFLBCJwBAEHhEJAEGg4BFQEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

   ' SelectedItem
   Add(4,
"gBFLBCJwBAEHhEJAEGg4BV4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK

   ' Marks a cell

Add(5,"gBFLBCJwBAEHhEJAEGg4BEcMQAAYAQGKIYBkAKBQAGaAoDDMOILQiMQxDPBMK


Add(6,"gBFLBCJwBAEHhEJAEGg4BaAFg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

}
BackColorHeader = 16777216          '0x01BBGGRR
BackColorSortBarCaption = 16777216     '0x01BBGGRR
BackColorSortBar = RGB(255,255,255)
FilterBarBackColor = 16777216        '0x01BBGGRR
Background(0) = 33554432             '0x02BBGGRR
Background(1) = 50331648             '0x03BBGGRR
Background(8) = 67108864             '0x04BBGGRR
```

```
Background(9) = 67108864            '0x04BBGGRR
Background(10) = 100663296            '0x06BBGGRR
Background(11) = 100663296            '0x06BBGGRR
Background(12) = 100663296            '0x06BBGGRR
Background(13) = 100663296            '0x06BBGGRR
Background(14) = 100663296            '0x06BBGGRR
SelBackColor = 67108864            '0x04BBGGRR
SelForeColor = 0
'ForeColorHeader = RGB(255,255,255)
'ForeColorSortBar = RGB(255,255,255)
ShowFocusRect = False

MarkSearchColumn = False
BackColor = RGB(255,255,255)
BackColorLevelHeader = RGB(255,255,255)
SortBarVisible = True
DrawGridLines = -1
ScrollBySingleLine = True
Columns
{
   "Name"
   {
      HeaderImage = 1
      DisplayFilterButton = True
      DisplayFilterDate = True
      Width = 180
   }
   "1"
   {
      HeaderImage  = 2
      HeaderImageAlignment = 2
      LevelKey = 1
      Def(0) = True
      Width = 18
      HTMLCaption = " 1  First"
   }
   "2"
```

```
   {
      HeaderImage  = 3
      HeaderImageAlignment = 2
      LevelKey = 1
      Def(0) = True
      Width = 18
      HTMLCaption = "2 Second"
   }
   "3"
   {
      HeaderImage  = 1
      HeaderImageAlignment = 2
      HTMLCaption = " 3  Third"
      LevelKey = 1
      Def(0) = True
      Width = 18
   }
   ""
   {
      LevelKey = 1
      Width = 40
      Def(1) = True
   }
}
Items
{
   Dim h
   h = Add("exListAdd an advanced List control to your application. ")
   CellMerge(h,0) = 1
   CellMerge(h,0) = 2
   CaptionFormat(h,0) = 1
   CellImage(h,0) = 1
   CellForeColor(h,1) = RGB(0,0,255)
   CellState(h,1) = 1
   CellSingleLine(h,0) = False
   CellToolTip(h,0) = "This is the cell's tooltip that should appear when the cursor is over
the cell.
```

```
cell's tooltip"
   CellPicture(h,0) =
"gBHJJGHA5MIqAAXAD3AENhozhpmhqZhrMhr/h0QGcQM0QTMQZkQf8QAESGcSM0STM



   h = Add("Item 2")
   CaptionFormat(h,0) = 1
   CellImage(h,0) = 3
   CellForeColor(h,1) = RGB(255,0,0)
   SelectItem(h) = True

   h = Add("Item 3")
   CaptionFormat(h,0) = 1
   CellImage(h,0) = 2
   CellForeColor(h,1) = RGB(255,0,0)

   h = Add("
The exList control supports features never seen in other controls.Exontrol
")
   CaptionFormat(h,0) = 1
   CellBackColor(h,0) = 83886080
   CellSingleLine(h,0) = False
   CellPicture(h,0) =
"gBHJJGHA5MIgAEIe4AAAFAoEDQXCoaEIdEkVi4lEgqEovEIVF8cF40F0jGw5FQdHI0EsrF0rk4

   CellMerge(h,0) = 1
   CellMerge(h,0) = 2
   CellMerge(h,0) = 3
   CellMerge(h,0) = 4
}
EndUpdate
```

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

Control/ClassName	Part	States
		CBS_UNCHECKED 1 CBS_UNCHECKE

BUTTON	BP_CHECKBOX = 3	CBS_UNCHECKED = 3 CBS_UNCHECKED = 4 CBS_CHECKED 5 CBS_CHECKEDH CBS_CHECKEDPR CBS_CHECKEDDIS CBS_MIXEDNORM CBS_MIXEDHOT = CBS_MIXEDPRESS CBS_MIXEDDISAB
	BP_GROUPBOX = 4	GBS_NORMAL = 1 GBS_DISABLED =
	BP_PUSHBUTTON = 1	PBS_NORMAL = 1 = 2 PBS_PRESSED PBS_DISABLED = PBS_DEFAULTED =
	BP_RADIOBUTTON = 2	RBS_UNCHECKED 1 RBS_UNCHECKE RBS_UNCHECKED = 3 RBS_UNCHECKED = 4 RBS_CHECKED 5 RBS_CHECKEDH RBS_CHECKEDPR RBS_CHECKEDDIS
	BP_USERBUTTON = 5	
CLOCK	CLP_TIME = 1	CLS_NORMAL = 1
COMBOBOX	CP_DROPDOWNBUTTON = 1	CBXS_NORMAL = CBXS_HOT = 2 CBXS_PRESSED = CBXS_DISABLED =
EDIT	EP_CARET = 2	
	EP_EDITTEXT = 1	ETS_NORMAL = 1 2 ETS_SELECTED ETS_DISABLED = ETS_FOCUSED = 5 ETS_READONLY = ETS_ASSIST = 7
EXPLORERBAR	EBP_HEADERBACKGROUND = 1	EBHC_NORMAL =

	EBP_HEADERCLOSE = 2	EBHC_HOT = 2 EBHC_PRESSED =
	EBP_HEADERPIN = 3	EBHP_NORMAL = EBHP_HOT = 2 EBHP_PRESSED = EBHP_SELECTEDN 4 EBHP_SELECTE EBHP_SELECTEDP 6
	EBP_IEBARMENU = 4	EBM_NORMAL = 1 = 2 EBM_PRESSE
	EBP_NORMALGROUPBACKGROUND = 5	
	EBP_NORMALGROUPCOLLAPSE = 6	EBNGC_NORMAL EBNGC_HOT = 2 EBNGC_PRESSED
	EBP_NORMALGROUPEXPAND = 7	EBNGE_NORMAL EBNGE_HOT = 2 EBNGE_PRESSED
	EBP_NORMALGROUPHEAD = 8	
	EBP_SPECIALGROUPBACKGROUND = 9	
	EBP_SPECIALGROUPCOLLAPSE = 10	EBSGC_NORMAL EBSGC_HOT = 2 EBSGC_PRESSED
	EBP_SPECIALGROUPEXPAND = 11	EBSGE_NORMAL EBSGE_HOT = 2 EBSGE_PRESSED
	EBP_SPECIALGROUPHEAD = 12	
HEADER	HP_HEADERITEM = 1	HIS_NORMAL = 1 2 HIS_PRESSED =
	HP_HEADERITEMLEFT = 2	HILS_NORMAL = 1 = 2 HILS_PRESSE
	HP_HEADERITEMRIGHT = 3	HIRS_NORMAL = 1 = 2 HIRS_PRESSE
	HP_HEADERSORTARROW = 4	HSAS_SORTEDUP HSAS_SORTEDDO
LISTVIEW	LVP_EMPTYTEXT = 5	
	LVP_LISTDETAIL = 3	
	LVP_LISTGROUP = 2	
		LIS_NORMAL = 1 2 LIS_SELECTED =

	LVP_LISTITEM = 1	LIS_DISABLED = 4 LIS_SELECTEDNO 5
	LVP_LISTSORTEDDETAIL = 4	
MENU	MP_MENUBARDROPDOWN = 4	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_MENUBARITEM = 3	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_CHEVRON = 5	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_MENUDROPDOWN = 2	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_MENUITEM = 1	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
	MP_SEPARATOR = 6	MS_NORMAL = 1 MS_SELECTED = 2 MS_DEMOTED = 3
MENUBAND	MDP_NEWAPPBUTTON = 1	MDS_NORMAL = 1 = 2 MDS_PRESSE MDS_DISABLED = MDS_CHECKED = MDS_HOTCHECKE
	MDP_SEPERATOR = 2	
PAGE	PGRP_DOWN = 2	DNS_NORMAL = 1 = 2 DNS_PRESSED DNS_DISABLED =
	PGRP_DOWNHORZ = 4	DNHZS_NORMAL = DNHZS_HOT = 2 DNHZS_PRESSED DNHZS_DISABLED
	PGRP_UP = 1	UPS_NORMAL = 1 = 2 UPS_PRESSED UPS_DISABLED =
	PGRP_UPHORZ = 3	UPHZS_NORMAL = UPHZS_HOT = 2 UPHZS_PRESSED

		UPHZS_DISABLED
PROGRESS	PP_BAR = 1	
	PP_BARVERT = 2	
	PP_CHUNK = 3	
	PP_CHUNKVERT = 4	
REBAR	RP_BAND = 3	
		CHEVS_NORMAL =
	RP_CHEVRON = 4	CHEVS_HOT = 2
		CHEVS_PRESSED
	RP_CHEVRONVERT = 5	
	RP_GRIPPER = 1	
	RP_GRIPPERVERT = 2	
		ABS_DOWNDISAB
		ABS_DOWNHOT,
		ABS_DOWNNORM
		ABS_DOWNPRESS
		ABS_UPDISABLED
		ABS_UPHOT,
		ABS_UPNORMAL,
SCROLLBAR	SBP_ARROWBTN = 1	ABS_UPPRESSED,
		ABS_LEFTDISABLI
		ABS_LEFTHOT,
		ABS_LEFTNORMA
		ABS_LEFTPRESSE
		ABS_RIGHTDISAB
		ABS_RIGHTHOT,
		ABS_RIGHTNORM
		ABS_RIGHTPRESS
	SBP_GRIPPERHORZ = 8	
	SBP_GRIPPERVERT = 9	
		SCRBS_NORMAL =
	SBP_LOWERTRACKHORZ = 4	SCRBS_HOT = 2
		SCRBS_PRESSED
		SCRBS_DISABLED
		SCRBS_NORMAL =
	SBP_LOWERTRACKVERT = 6	SCRBS_HOT = 2
		SCRBS_PRESSED
		SCRBS_DISABLED
		SCRBS_NORMAL =
		SCRBS_HOT = 2

	SBP_THUMBBTNHORZ = 2	SCRBS_PRESSED SCRBS_DISABLED
	SBP_THUMBBTNVERT = 3	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_UPPERTRACKHORZ = 5	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_UPPERTRACKVERT = 7	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_SIZEBOX = 10	SZB_RIGHTALIGN SZB_LEFTALIGN =
SPIN	SPNP_DOWN = 2	DNS_NORMAL = 1 = 2 DNS_PRESSED DNS_DISABLED =
	SPNP_DOWNHORZ = 4	DNHZS_NORMAL = DNHZS_HOT = 2 DNHZS_PRESSED DNHZS_DISABLED
	SPNP_UP = 1	UPS_NORMAL = 1 = 2 UPS_PRESSED UPS_DISABLED =
	SPNP_UPHORZ = 3	UPHZS_NORMAL = UPHZS_HOT = 2 UPHZS_PRESSED UPHZS_DISABLED
STARTPANEL	SPP_LOGOFF = 8	
	SPP_LOGOFFBUTTONS = 9	SPLS_NORMAL = SPLS_HOT = 2 SPLS_PRESSED =
	SPP_MOREPROGRAMS = 2	
	SPP_MOREPROGRAMSARROW = 3	SPS_NORMAL = 1 = 2 SPS_PRESSED
	SPP_PLACESLIST = 6	
	SPP_PLACESLISTSEPARATOR = 7	
	SPP_PREVIEW = 11	

	SPP_PROGLIST = 4	
	SPP_PROGLISTSEPARATOR = 5	
	SPP_USERPANE = 1	
	SPP_USERPICTURE = 10	
STATUS	SP_GRIPPER = 3	
	SP_PANE = 1	
	SP_GRIPPERPANE = 2	
TAB	TABP_BODY = 10	
	TABP_PANE = 9	
	TABP_TABITEM = 1	TIS_NORMAL = 1 T 2 TIS_SELECTED = TIS_DISABLED = 4 TIS_FOCUSED = 5
	TABP_TABITEMBOTHEDGE = 4	TIBES_NORMAL = TIBES_HOT = 2 TIBES_SELECTED TIBES_DISABLED TIBES_FOCUSED =
	TABP_TABITEMLEFTEDGE = 2	TILES_NORMAL = TILES_HOT = 2 TILES_SELECTED TILES_DISABLED = TILES_FOCUSED =
	TABP_TABITEMRIGHTEDGE = 3	TIRES_NORMAL = TIRES_HOT = 2 TIRES_SELECTED TIRES_DISABLED TIRES_FOCUSED
	TABP_TOPTABITEM = 5	TTIS_NORMAL = 1 = 2 TTIS_SELECTE TTIS_DISABLED = TTIS_FOCUSED =
	TABP_TOPTABITEMBOTHEDGE = 8	TTIBES_NORMAL TTIBES_HOT = 2 TTIBES_SELECTE TTIBES_DISABLED TTIBES_FOCUSED
	TABP_TOPTABITEMLEFTEDGE = 6	TTILES_NORMAL = TTILES_HOT = 2 TTILES_SELECTE

		TTILES_DISABLED
		TTILES_FOCUSED
		TTIRES_NORMAL
		TTIRES_HOT = 2
	TABP_TOPTABITEMRIGHTEDGE = 7	TTIRES_SELECTE
		TTIRES_DISABLED
		TTIRES_FOCUSED
TASKBAND	TDP_GROUPCOUNT = 1	
	TDP_FLASHBUTTON = 2	
	TDP_FLASHBUTTONGROUPMENU = 3	
TASKBAR	TBP_BACKGROUNDBOTTOM = 1	
	TBP_BACKGROUNDLEFT = 4	
	TBP_BACKGROUNDRIGHT = 2	
	TBP_BACKGROUNDTOP = 3	
	TBP_SIZINGBARBOTTOM = 5	
	TBP_SIZINGBARBOTTOMLEFT = 8	
	TBP_SIZINGBARRIGHT = 6	
	TBP_SIZINGBARTOP = 7	
		TS_NORMAL = 1 T
		TS_PRESSED = 3
TOOLBAR	TP_BUTTON = 1	TS_DISABLED = 4
		TS_CHECKED = 5
		TS_HOTCHECKED
		TS_NORMAL = 1 T
		TS_PRESSED = 3
	TP_DROPDOWNBUTTON = 2	TS_DISABLED = 4
		TS_CHECKED = 5
		TS_HOTCHECKED
		TS_NORMAL = 1 T
		TS_PRESSED = 3
	TP_SPLITBUTTON = 3	TS_DISABLED = 4
		TS_CHECKED = 5
		TS_HOTCHECKED
		TS_NORMAL = 1 T
		TS_PRESSED = 3
	TP_SPLITBUTTONDROPDOWN = 4	TS_DISABLED = 4
		TS_CHECKED = 5
		TS_HOTCHECKED
		TS_NORMAL = 1 T
		TS_PRESSED = 3

	TP_SEPARATOR = 5	TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
		TS_NORMAL = 1 T TS_PRESSED = 3
	TP_SEPARATORVERT = 6	TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
TOOLTIP	TTP_BALLOON = 3	TTBS_NORMAL = TTBS_LINK = 2
	TTP_BALLOONTITLE = 4	TTBS_NORMAL = TTBS_LINK = 2
	TTP_CLOSE = 5	TTCS_NORMAL = TTCS_HOT = 2 TTCS_PRESSED =
	TTP_STANDARD = 1	TTSS_NORMAL = TTSS_LINK = 2
	TTP_STANDARDTITLE = 2	TTSS_NORMAL = TTSS_LINK = 2
TRACKBAR	TKP_THUMB = 3	TUS_NORMAL = 1 2 TUS_PRESSED = TUS_FOCUSED = 4 TUS_DISABLED =
	TKP_THUMBBOTTOM = 4	TUBS_NORMAL = TUBS_HOT = 2 TUBS_PRESSED = TUBS_FOCUSED = TUBS_DISABLED =
	TKP_THUMBLEFT = 7	TUVLS_NORMAL = TUVLS_HOT = 2 TUVLS_PRESSED TUVLS_FOCUSED TUVLS_DISABLED
	TKP_THUMBRIGHT = 8	TUVRS_NORMAL = TUVRS_HOT = 2 TUVRS_PRESSED TUVRS_FOCUSED TUVRS_DISABLED
		TUTS_NORMAL = TUTS_HOT = 2

	TKP_THUMBTOP = 5	TUTS_PRESSED = TUTS_FOCUSED = TUTS_DISABLED =
	TKP_THUMBVERT = 6	TUVS_NORMAL = TUVS_HOT = 2 TUVS_PRESSED = TUVS_FOCUSED = TUVS_DISABLED =
	TKP_TICS = 9	TSS_NORMAL = 1
	TKP_TICSVERT = 10	TSVS_NORMAL =
	TKP_TRACK = 1	TRS_NORMAL = 1
	TKP_TRACKVERT = 2	TRVS_NORMAL =
TRAYNOTIFY	TNP_ANIMBACKGROUND = 2	
	TNP_BACKGROUND = 1	
TREEVIEW	TVP_BRANCH = 3	
	TVP_GLYPH = 2	GLPS_CLOSED = GLPS_OPENED =
	TVP_TREEITEM = 1	TREIS_NORMAL = TREIS_HOT = 2 TREIS_SELECTED TREIS_DISABLED TREIS_SELECTED = 5
WINDOW	WP_CAPTION = 1	CS_ACTIVE = 1 CS = 2 CS_DISABLED
	WP_CAPTIONSIZINGTEMPLATE = 30	
	WP_CLOSEBUTTON = 18	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
	WP_DIALOG = 29	
	WP_FRAMEBOTTOM = 9	FS_ACTIVE = 1 FS = 2
	WP_FRAMEBOTTOMSIZINGTEMPLATE = 36	
	WP_FRAMELEFT = 7	FS_ACTIVE = 1 FS = 2
	WP_FRAMELEFTSIZINGTEMPLATE = 32	
	WP_FRAMERIGHT = 8	FS_ACTIVE = 1 FS = 2
	WP_FRAMERIGHTSIZINGTEMPLATE = 34	

WP_HELPBUTTON = 23	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_HORZSCROLL = 25	HSS_NORMAL = 1 = 2 HSS_PUSHED HSS_DISABLED =
WP_HORZTHUMB = 26	HTS_NORMAL = 1 2 HTS_PUSHED = HTS_DISABLED =
WP_MAX_BUTTON	MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_MAXCAPTION = 5	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
WP_MDICLOSEBUTTON = 20	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
WP_MDIHELPBUTTON = 24	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_MDIMINBUTTON = 16	MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MDIRESTOREBUTTON = 22	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_MDISYSBUTTON = 14	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_MINBUTTON = 15	MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MINCAPTION = 3	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_RESTOREBUTTON = 21	RBS_NORMAL = 1 = 2 RBS_PUSHED

WP_SMALLCAPTION = 2	RBS_DISABLED = CS_ACTIVE = 1 CS = 2 CS_DISABLED
WP_SMALLCAPTIONSIZINGTEMPLATE = 31	
WP_SMALLCLOSEBUTTON = 19	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
WP_SMALLFRAMEBOTTOM = 12	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMEBOTTOMSIZINGTEMPLATE = 37	
WP_SMALLFRAMELEFT = 10	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMELEFTSIZINGTEMPLATE = 33	
WP_SMALLFRAMERIGHT = 11	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMERIGHTSIZINGTEMPLATE = 35	
WP_SMALLHELPBUTTON	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_SMALLMAXBUTTON	MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_SMALLMAXCAPTION = 6	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
WP_SMALLMINCAPTION = 4	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_SMALLRESTOREBUTTON	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_SMALLSYSBUTTON	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_SYSBUTTON = 13	SBS_NORMAL = 1 = 2 SBS_PUSHED

	SBS_DISABLED =
WP_VERTSCROLL = 27	VSS_NORMAL = 1 = 2 VSS_PUSHED VSS_DISABLED =
WP_VERTTHUMB = 28	VTS_NORMAL = 1 2 VTS_PUSHED = VTS_DISABLED =

method Appearance.Clear ()

Removes all skins in the control.

Type	Description

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

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

- control's border, Appearance property
- control's **header bar**, BackColorHeader property
- control's **filter bar**, FilterBarBackColor property
- control's **sort bar**, BackColorSort property
- the caption of the control's sort bar, BackColorSortCaption property
- **selected item** or cell, SelBackColor property
- **item**, ItemBackColor property
- **cell**, CellBackColor property
- cell's **button**, "**drop down**" filter bar button, "close" filter bar button, tooltips, and so on, Background property
- CellImage, CellImages, HeaderImage, CheckImage or RadioImage 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:

- control's border, Appearance property
- control's **header bar**, BackColorHeader property
- control's **filter bar**, FilterBarBackColor property
- control's **sort bar**, BackColorSort property
- the caption of the control's sort bar, BackColorSortCaption property
- **selected item** or cell, SelBackColor property
- **item**, ItemBackColor property
- **cell**, CellBackColor property
- cell's **button**, "**drop down**" filter bar button, "close" filter bar button, tooltips, and so on, Background property
- CellImage, CellImages, HeaderImage, CheckImage or RadioImage property

property Appearance.RenderType as Long

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

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

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

The RenderType property is applied to all parts that displays an EBN object. The properties of color type may support the EBN object if the property's description includes "*A color expression that indicates the cell's background color. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.*" In other words, a property that supports EBN objects should be of format 0xIDRRGGBB, where the ID is the identifier of the EBN to be applied, while the BBGGRR is the (Red,Green,Blue, RGB-Color) color to be applied on the selected EBN. For instance, the 0x1000000 indicates displaying the EBN as it is, with no color applied, while the 0x1FF0000, applies the Blue color (RGB(0x0,0x0,0xFF), RGB(0,0,255) on the EBN with the identifier 1. You can use the EBNColor tool to visualize applying EBN colors.

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

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

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

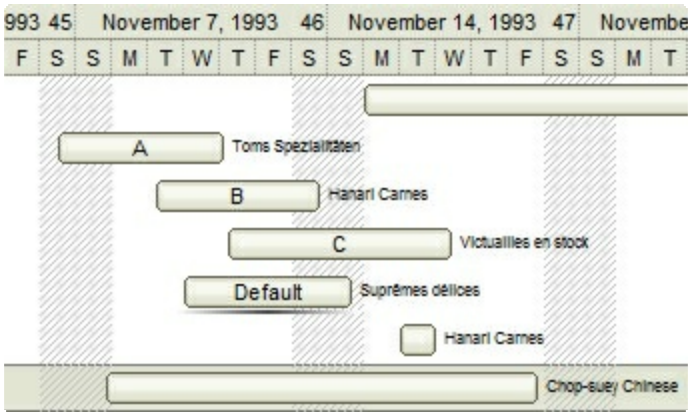
In the following screen shot the following objects displays the current EBN with a different color:

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

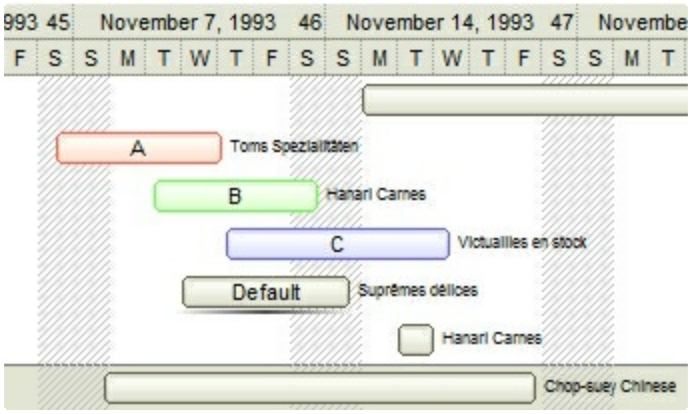
- "C" in Blue (RGB(0,0,255), for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

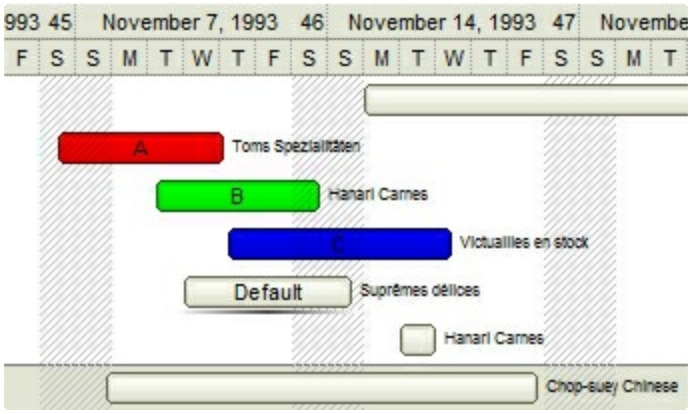
- ***-3**, no color is applied*. For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.



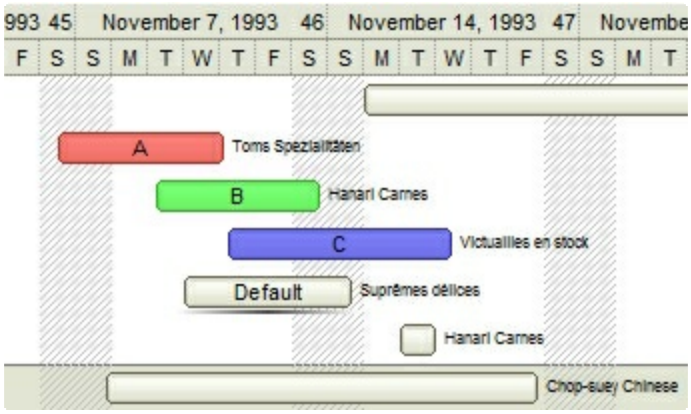
- ***-2**, OR-color scheme*. The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the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



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

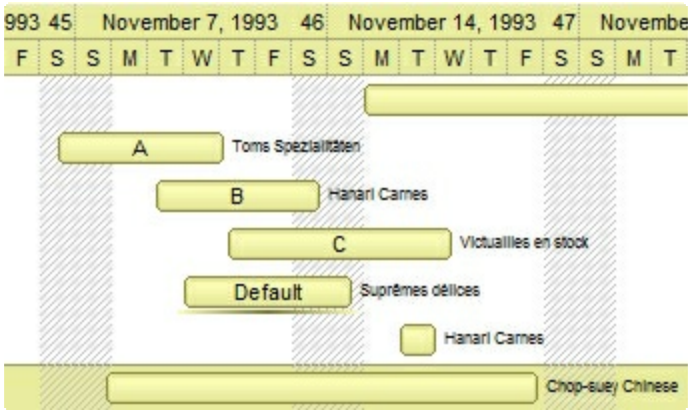


- ***0**, default*, the specified color is applied to the EBN. For instance, the BackColorHeader = &H1FF0000, applies a Blue color to the object. This option could be used to specify any color for the part of the components, that support EBN objects, not only solid colors.

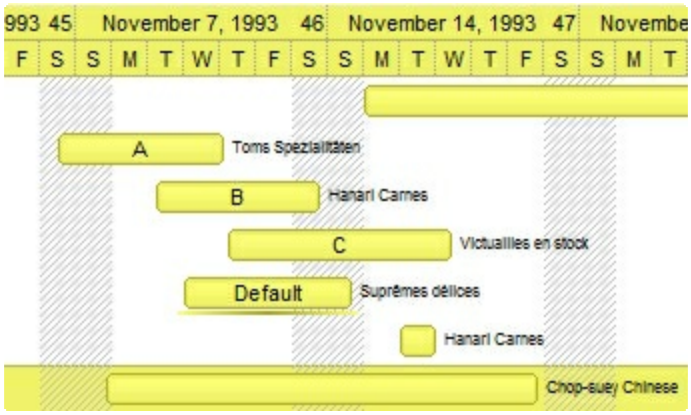


- **0xAABBGGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on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

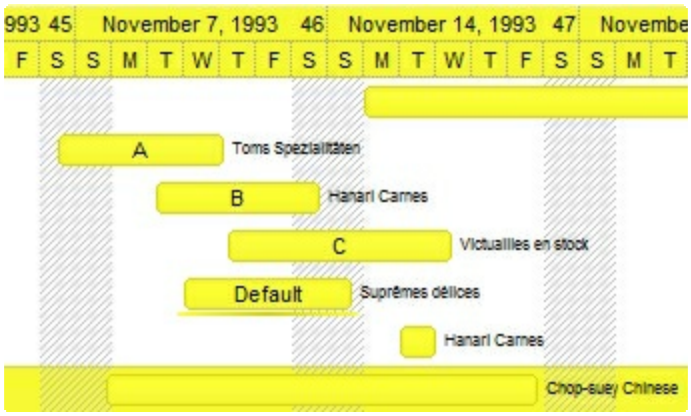
*The following picture shows the control with the RenderType property on 0x**40**00FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256):*



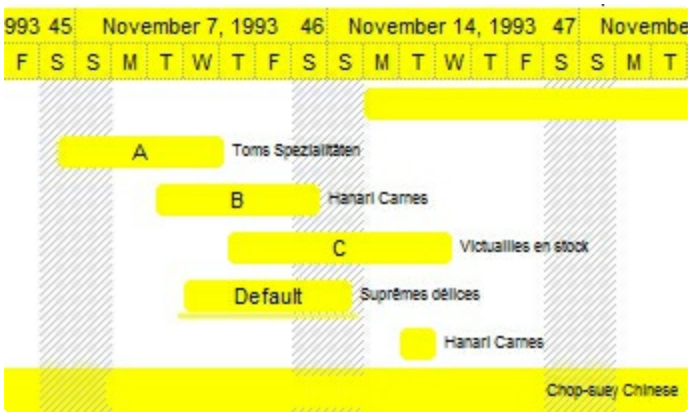
*The following picture shows the control with the RenderType property on 0x**80**00FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256):*



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



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


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

Column object

The ExList control supports multiple columns. The Columns object contains a collection of Column objects. By default, the control doesn't add any default column, so the user has to add at least one column, before inserting any new items. The Column object holds information about a column. The Column object supports the following properties:

Name	Description
Alignment	Specifies the column's alignment.
AllowDragging	Retrieves or sets a value indicating whether the user will be able to drag the column.
AllowSizing	Retrieves or sets a value indicating whether the user will be able to change the width of the visible column by dragging.
AllowSort	Returns or sets a value that indicates whether the user can sort the column by clicking the column's header.
AutoSearch	Specifies the kind of searching while user types characters within the columns.
AutoWidth	Computes the column's width required to fit the entire column's content.
Caption	Retrieves or sets a value that indicates the column's caption.
ComputedField	Retrieves or sets a value that indicates the formula of the computed column.
CustomFilter	Retrieves or sets a value that indicates the list of custom filters.
Data	Associates an extra data to the column.
Def	Retrieves or sets a value that indicates the default value of given properties for all cells in the same column.
DefaultSortOrder	Specifies whether the default sort order is ascending or descending.
DisplayFilterButton	Specifies whether the column's header displays the filter button.
DisplayFilterDate	Specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for.
DisplayFilterPattern	Specifies whether the dropdown filter bar contains a textbox for editing the filter as pattern.

DisplaySortIcon	Retrieves or sets a value indicating whether the sort icon is visible on column's header, while the column is sorted.
Enabled	Returns or sets a value that determines whether a column's header can respond to user-generated events.
Filter	Specifies the column's filter when filter type is exFilter, exPattern or exDate.
FilterBarDropDownWidth	Specifies the width of the drop down filter window proportionally with the width of the column.
FilterList	Specifies whether the drop down filter list includes visible or all items.
FilterOnType	Filters the column as user types characters in the drop down filter window.
FilterType	Specifies the column's filter type.
FireFormatColumn	Retrieves or sets a value that indicates whether the control fires the FormatColumn event in order to format the caption for each cell in the column.
FormatColumn	Specifies the format to display the cells in the column.
HeaderAlignment	Specifies the alignment of the column's caption.
HeaderBold	Retrieves or sets a value that indicates whether the column's caption is bolded.
HeaderImage	Retrieves or sets a value indicating the index of an Image in the Images collection, that is displayed in the column's header.
HeaderImageAlignment	Retrieves or sets the alignment of the image into the column's header.
HeaderItalic	Retrieves or sets a value that indicates whether the column's caption should appear in italic.
HeaderStrikeOut	Retrieves or sets a value that indicates whether the column's caption should appear in strikeout.
HeaderUnderline	Retrieves or sets a value that indicates whether the column's caption is underlined.
HeaderVertical	Specifies whether the column's header is vertically displayed.
HTMLCaption	Retrieves or sets the text in HTML format displayed in the column's header.
Index	Returns a value that represents the index of an object in a

	collection.
Key	Retrieves or sets a the column's key.
LevelKey	Retrieves or sets a value that indicates the key of the column's level.
MaxWidthAutoResize	Retrieves or sets a value that indicates the maximum column's width when the WidthAutoResize is True.
MinWidthAutoResize	Retrieves or sets a value that indicates the minimum column's width when the WidthAutoResize is True.
Position	Retrieves or sets a value that indicates the position of the column in the header bar.
ShowFilter	Shows the column's filter window.
SortOrder	Specifies the column's sort order.
SortPosition	Returns or sets a value that indicates the position of the column in the sorting columns collection.
SortType	Returns or sets a value that indicates the way a control sorts the values for a column.
ToolTip	Specifies the column's tooltip description.
Visible	Retrieves or sets a value indicating whether the column is visible or hidden.
Width	Retrieves or sets a value that determines the column's width.
WidthAutoResize	Retrieves or sets a value that indicates whether the column is automatically resized according to the width of the contents within the column.

property Column.Alignment as AlignmentEnum

Retrieves or sets the alignment of the caption into the column's header.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the cells in the column.

The Alignment property aligns the cells in the column. The HeaderAlignment property aligns the caption of the column in the column's header. Use the CellHAlignment property to align a particular cell. By default, all columns are aligned to the left. Use the CellVAlignment property to align vertically the cell's caption.

property Column.AllowDragging as Boolean

Retrieves or sets a value indicating whether the user will be able to drag the column.

Type	Description
Boolean	A boolean expression indicating whether the user is able to drag the column.

By default, the AllowDragging property is True. The AllowDragging property specifies whether the user can click the control's header and drag it to another position. Use the AllowSizing property to specify whether the user can resize the column at runtime. Use the Position property to change pragmatically the column's position. Use the AllowSort property to avoid sorting a column when the user clicks the column's header.

property Column.AllowSizing as Boolean

Retrieves or sets a value indicating whether the user will be able to change the width of the visible column by dragging.

Type	Description
Boolean	A boolean expression that indicates whether the user will be able to change the width of the visible columns by dragging.

By default, the AllowSizing property is True. Use the AllowSizing property to fix the column's width. Use the ColumnAutoResize property of the control to fit the visible columns to the control's client area. Use the AllowSort property to avoid sorting a column when the user clicks the column's header. Use the AllowDragging property to specify whether the user can click the column's header and drag it to another position. Use the Width property to specify the column's width, at runtime. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible.

property Column.AllowSort as Boolean

Returns or sets a value that indicates whether the user can sort the column by clicking the column's header.

Type	Description
Boolean	A boolean expression that indicates whether the column gets sorted when the user clicks the column's header.

Sorting by a single column in the control is a simple matter of clicking on the column head. Sorting by multiple columns, however, is not so obvious. But it's actually quite easy. First, sort by the first criterion, by clicking on the column head. Then hold the Shift key down as you click on a second heading. Another option is dragging the column's header to the control's sort bar. The SortBarVisible property shows the control's sort bar. Use the AllowSort property to avoid sorting a column when the user clicks the column's header. Use the SortOnClick property to specify the action that control executes when the user clicks the column's head. The control fires the Sort event when the control sorts a column (the user clicks the column's head) or when the sorting position is changed in the control's sort bar. Use the AllowDragging property to specify whether the column's header can be dragged. Use the DefaultSortOrder property to specify the column's default sort order, when the user first clicks the column's header.

property Column.AutoSearch as AutoSearchEnum

Specifies the kind of searching while user types characters within the columns.

Type	Description
AutoSearchEnum	An AutoSearchEnum expression that defines the type of incremental searching.

By default, the AutoSearch property is exStartWith. The AutoSearch property has effect only if the AutoSearch property of the control is True. Use the AutoSearch property to define a 'contains' incremental search.

- If the AutoSearch property is exContains, the control searches for items that contains the typed characters.
- If the AutoSearch property is exStartWith, the control searches for items that starts with the typed characters.

The searching column is defined by the SearchColumnIndex property.

property Column.AutoWidth as Long

Computes the column's width required to fit the entire column's content.

Type	Description
Long	A long value that indicates the required width of the column to fit the entire column's content.

Use the AutoWidth property to arrange the columns to fit the entire control's content. The AutoWidth property scans all cells of the column. The AutoWidth property. Use the Width property to change the column's width at runtime. Use the WidthAutoResize property to let control resizes the column each time when an item is expanded or collapsed. Use the ColumnAutoResize property to specify whether the control resizes all visible columns to fit the control's client area.

The following VB function resizes all columns:

```
Private Sub autoSize(ByVal t As EXLISTLibCtl.List)
   t.BeginUpdate
      Dim c As Column
      For Each c In t.Columns
         c.Width = c.AutoWidth
      Next
      t.EndUpdate
      t.Refresh
End Sub
```

The following C++ sample resizes all visible columns:

```
#include "Columns.h"
#include "Column.h"
void autoSize( CList& list )
{
   list.BeginUpdate();
   CColumns columns = list.GetColumns();
   for ( long i = 0;i < columns.GetCount(); i++ )
   {
      CColumn column = columns.GetItem( COleVariant( i ) );
      if ( column.GetVisible() )
         column.SetWidth( column.GetAutoWidth() );
```

```
    }
    list.EndUpdate();
}
```

The following VB.NET sample resizes all visible columns:

```
Private Sub autoSize(ByRef list As AxEXLISTLib.AxList)
    list.BeginUpdate()
    Dim i As Integer
    With list.Columns
        For i = 0 To .Count - 1
            If .Item(i).Visible Then
                .Item(i).Width = .Item(i).AutoWidth
            End If
        Next
    End With
    list.EndUpdate()
End Sub
```

The following C# sample resizes all visible columns:

```
private void autoSize( ref AxEXLISTLib.AxList list )
{
    list.BeginUpdate();
    for ( int i = 0; i < list.Columns.Count - 1; i++ )
        if ( list.Columns[i].Visible)
            list.Columns[i].Width = list.Columns[i].AutoWidth;
    list.EndUpdate();
}
```

The following VFP sample resizes all visible columns:

```
with thisform.List1
    .BeginUpdate()
    for i = 0 to .Columns.Count - 1
        if ( .Columns(i).Visible )
            .Columns(i).Width = .Columns(i).AutoWidth
        endif
    next
```

```
    .EndUpdate()
endwith
```

property Column.Caption as String

Retrieves or sets a value that indicates the column's caption.

Type	Description
String	A string expression that indicates the column's caption.

Each property of the Items object that has an argument ColIndex can use the column's caption to identify a column. Adding two columns with the same caption is accepted and these are differentiated by their indexes. To hide a column use the Visible property of the Column object. The column's caption is displayed using the following font attributes: HeaderBold, HeaderItalic, HeaderUnderline, HeaderStrikeout. Use the HTMLCaption property to specify the column's caption using built-in HTML tags. Use the Add method to add new columns and to specify their captions.

property Column.ComputedField as String

Retrieves or sets a value that indicates the formula of the computed column.

Type	Description
String	A String expression that indicates the formula to compute the field/cell. The formula is applied to all cells in the column with the CaptionFormat property on exText (the exText value is by default).

A computed field or cell displays the result of an arithmetic formula that may include operators, variables and constants. By default, the ComputedField property is empty. If the the ComputedField property is empty, the property have no effect. If the ComputedField property is not empty, all cells in the column, that have the CaptionFormat property on exText, uses the same formula to display their content. For instance, you can use the CaptionFormat property on exHTML, for cells in the column, that need to display other things than column's formula, or you can use the CaptionFormat property on exComputedField, to change the formula for a particular cell. Use the FormatColumn property to format the column. Use the CaptionFormat property to change the type for a particular cell. Use the Caption property to specify the cell's content. For instance, if the CaptionFormat property is exComputedField, the Caption property indicates the formula to compute the cell's content. The Def(exCaptionFormat) property is changed to exComputedField, each time the ComputeField property is changed to a not empty value. If the ComputedField property is set to an empty string, the Def(exCaptionFormat) property is set to exText. Call the Refresh method to force refreshing the control.

The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The Caption property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*

This property/method supports predefined constants and operators/functions as described here.

Samples:

1. "**1**", the cell displays 1
2. "**%0 + %1**", the cell displays the sum between cells in the first and second columns.
3. "**%0 + %1 - %2**", the cell displays the sum between cells in the first and second columns minus the third column.
4. "**(%0 + %1)*0.19**", the cell displays the sum between cells in the first and second columns multiplied with 0.19.

5. "**(%0 + %1 + %2)/3**", the cell displays the arithmetic average for the first three columns.
6. "**%0 + %1 < %2 + %3**", displays 1 if the sum between cells in the first two columns is less than the sum of third and forth columns.
7. "**proper(%0)**'" formats the cells by capitalizing first letter in each word
8. "**currency(%1)**'" displays the second column as currency using the format in the control panel for money
9. "**len(%0) ? currency(dbl(%0)) : ""**" displays the currency only for not empty/blank cells.
10. "**int(date(%1)-date(%2)) + 'D ' + round(24*(date(%1)-date(%2) - floor(date(%1)-date(%2)))) + 'H'**"" displays interval between two dates in days and hours, as xD yH
11. **"2:=((1:=int(0:= date(%1)-date(%0))) = 0 ? '' : str(=:1) + ' day(s)') + (3:=round(24*(=:0-floor(=:0))) ? (len(=:2) ? ' and ' : '') + =:3 + ' hour(s)' : '')"** displays the interval between two dates, as x day(s) [and y hour(s)], where the x indictaes the number of days, and y the number of hours. The hour part is missing, if 0 hours is displayed, or nothing is displayed if dates are identical.

property Column.CustomFilter as String

Retrieves or sets a value that indicates the list of custom filters.

Type	Description
String	A String expression that defines the list of custom filters.

By default, the CustomFilter property is empty. The CustomFilter property has effect only if it is not empty, and the FilterType property is not exImage, exCheck or exNumeric. Use the DisplayFilterPattern property to hide the text box to edit the pattern, in the drop down filter window. The All predefined item and the list of custom filter is displayed in the drop down filter window, if the CustomFilter property is not empty. The Blanks and NonBlanks predefined items are not defined, when custom filter is displayed. Use the Description(exFilterBarAll) property on empty string to hide the All predefined item, in the drop down filter window. Use the DisplayFilterButton property to show the button on the column's header to drop down the filter window. Use the Background property to define the visual appearance for the drop down button.

The CustomFilter property defines the list of custom filters as pairs of (caption,pattern) where the caption is displayed in the drop down filter window, and the pattern is get selected when the user clicks the item in the drop down filter window (the FilterType property is set on exPattern, and the Filter property defines the custom pattern being selected). The caption and the pattern are separated by a "||" string (two vertical bars, character 124). The pattern expression may contains multiple patterns separated by a single "|" character (vertical bar, character 124). A pattern may contain the wild card characters '?' for any single character, '*' for zero or more occurrences of any character, '#' for any digit character. If any of the *, ?, # or | characters are preceded by a \ (escape character) it masks the character itself. If the pattern is not present in the (caption,pattern) pair, the caption is considered as being the pattern too. The pairs in the list of custom patterns are separated by "|||" string (three vertical bars, character 124). So, the syntax of the CustomFilter property should be of: CAPTION [**||** PATTERN [**|** PATTERN]] [**|||** CAPTION [**||** PATTERN [**|** PATTERN]]].

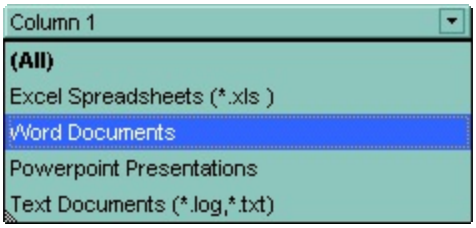
For example, you may have a list of documents and instead of listing the name of each document in the filter drop down list for the names column you may want to list the following:

Excel Spreadsheets
Word Documents
Powerpoint Presentations
Text Documents

And define the filter patterns for each line above as follows:

*.xls
*.doc
*.pps
*.txt, *.log

and so the CustomFilter property should be "**Excel Spreadsheets (*.xls)||*.xls|||Word Documents||*.doc|||Powerpoint Presentations||*.pps|||Text Documents (*.log,*.txt)||*.txt|*.log**". The following screen shot shows this custom filter format



property Column.Data as Variant

Associates an extra data to the column.

Type	Description
Variant	A Variant expression that indicates the column's extra data.

Use the Data property to assign any extra data to a column. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item

property Column.Def(Property as DefColumnEnum) as Variant

Retrieves or sets a value that indicates the default value of given properties for all cells in the same column.

Type	Description
Property as DefColumnEnum	A DefColumnEnum expression that indicates the property being changed.
Variant	A Variant value that specifies the newly value.

Use the Def property to specify a common value for given properties for all cells in the column. For instance, you can use the Def property to assign check boxes to all cells in the column, without enumerating them. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample assigns checkboxes for all cells in the first column:

```
List1.Columns(0).Def(exCellHasCheckBox) = True
```

The following VB sample changes the background color for all cells in the first column:

```
List1.Columns(0).Def(exCellBackColor) = RGB(240, 240, 240)
```

The following C++ sample assigns checkboxes for all cells in the first column:

```
COleVariant vtCheckBox( VARIANT_TRUE );
m_list.GetColumns().GetItem( COleVariant( (long) 0 ) ).SetDef( /*exCellHasCheckBox*/ 0,
vtCheckBox );
```

The following C++ sample changes the background color for all cells in the first column:

```
COleVariant vtBackColor( (long)RGB(240, 240, 240) );
m_list.GetColumns().GetItem( COleVariant( (long) 0 ) ).SetDef( /*exCellBackColor*/ 4,
vtBackColor );
```

The following VB.NET sample assigns checkboxes for all cells in the first column:

```
With AxList1.Columns(0)
   .Def(EXLISTLib.DefColumnEnum.exCellHasCheckBox) = True
End With
```

The following VB.NET sample changes the background color for all cells in the first column:

```
With AxList1.Columns(0)
   .Def(EXLISTLib.DefColumnEnum.exCellBackColor) = ToUInt32(Color.WhiteSmoke)
End With
```

where the ToUInt32 function converts a Color expression to OLE_COLOR,

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
   Dim i As Long
   i = c.R
   i = i + 256 * c.G
   i = i + 256 * 256 * c.B
   ToUInt32 = Convert.ToUInt32(i)
End Function
```

The following C# sample assigns checkboxes for all cells in the first column:

```
axList1.Columns[0].set_Def( EXLISTLib.DefColumnEnum.exCellHasCheckBox, true );
```

The following C# sample changes the background color for all cells in the first column:

```
axList1.Columns[0].set_Def(EXLISTLib.DefColumnEnum.exCellBackColor,
ToUInt32(Color.WhiteSmoke));
```

where the ToUInt32 function converts a Color expression to OLE_COLOR,

```
private UInt32 ToUInt32(Color c)
{
   long i;
   i = c.R;
   i = i + 256 * c.G;
   i = i + 256 * 256 * c.B;
   return Convert.ToUInt32(i);
}
```

The following VFP sample assigns checkboxes for all cells in the first column:

```
with thisform.List1.Columns(0)
   .Def( 0 ) = .t.
```

```
endwith
```

The following VFP sample changes the background color for all cells in the first column:

```
with thisform.List1.Columns(0)
   .Def( 4 ) = RGB(240,240,240)
endwith
```

property Column.DefaultSortOrder as Boolean

Specifies whether the default sort order is ascending or descending.

Type	Description
Boolean	A boolean expression that specifies the default sort order.

Use the DefaultSortOrder property to specify the default sort order, when the column's header is clicked. Use the SortOnClick property to specify the action that control takes when the user clicks the column's header. The SortOrder property specifies the column's sort order. Use the Sort method to sort items at runtime. Use the SingleSort property to allow sorting by multiple columns.

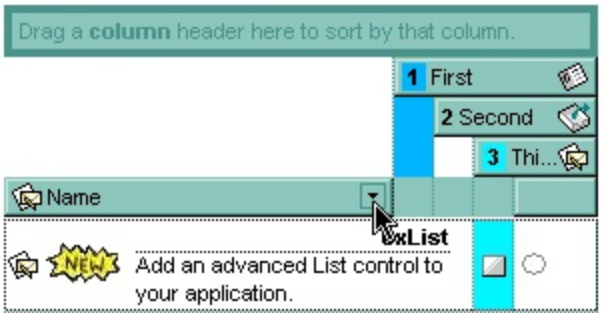
property Column.DisplayFilterButton as Boolean

Shows or hides the column's filter bar button.

Type	Description
Boolean	A boolean expression that indicates whether the column's filter bar button is visible or hidden.

By default, the DisplayFilterButton property is False. The column's filter button is displayed on the column's caption. *Use the FilterOnType property to enable the Filter-On-Type feature, that allows you to filter the control's data based on the characters you type.*

The DisplayFilterPattern property determines whether the column's filter window includes the "Filter For" (pattern) field. Use the DisplayFilterDate property to include a date selector to the column's drop down filter window. Use the FilterBarDropDownHeight property to specify the height of the drop down filter window. Use the FilterType property to specify the type of the column's filter. Use the FilterList property to specify the list of items being included in the column's drop down filter list. Use FilterBarDropDownWidth property to specify the width of the drop down filter window. Use the Background property to change the visual appearance for the drop down filter button. Use the FilterCriteria property to specify the filter criteria usinr OR, AND or NOT operators. Use the CustomFilter property to define you custom filters. Use the ShowFilter method to show programmatically the column's drop down filter window.

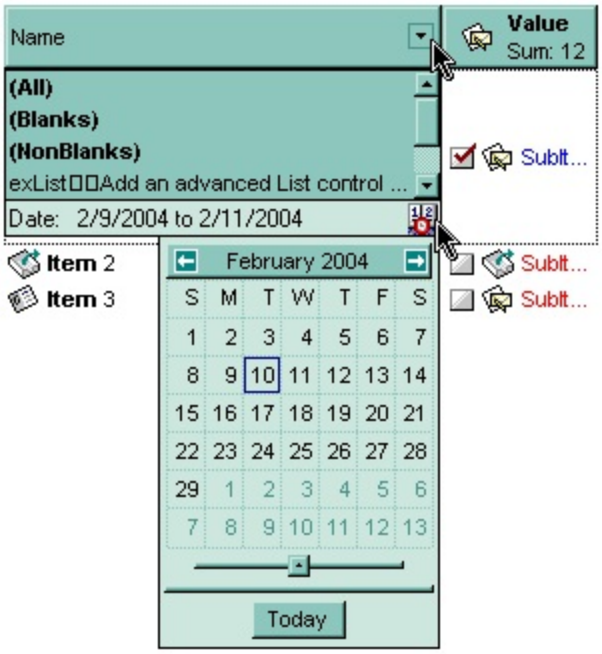


property Column.DisplayFilterDate as Boolean

Specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for.

Type	Description
Boolean	A boolean expression that indicates whether the drop down filter window displays a date selector to filter items into a given interval.

By default, the DisplayFilterDate property is False. Use the DisplayFilterDate property to filter items that match a given interval of dates. The DisplayFilterDate property includes a date button to the right of the Date field in the drop down filter window. The DisplayFilterDate property has effect only if the DisplayFilterPattern property is True. If the user clicks the filter's date selector the control displays a built-in calendar editor to help user to include a date to the date field of the drop down filter window. Use the Description property to customize the strings being displayed on the drop down filter window. If the Date field in the filter drop down window is not empty, the FilterType property of the Column object is set on exDate, and the Filter property of the Column object points to the interval of dates being used when filtering.



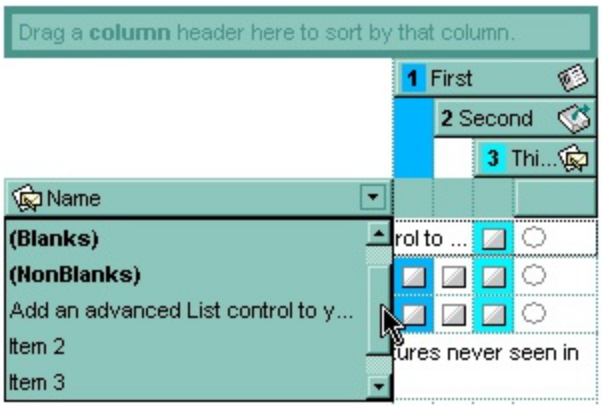
property Column.DisplayFilterPattern as Boolean

Specifies whether the dropdown filter bar contains a textbox for editing the filter as pattern.

Type	Description
Boolean	A boolean expression that indicates whether the pattern field is visible or hidden.

Use the DisplayFilterButton property to show the column's filter button. If the DisplayFilterButton property is False the drop down filter window doesn't include the "Filter For" or "Date" field. Use the DisplayFilterDate property to filter items that match a given interval of dates. Use the CustomFilter property to define you custom filters. The "Filter For" (pattern) field in the drop down filter window is always shown if the FilterOnType property is True, no matter of the DisplayFilterPattern property.

The drop down filter window displays the "Filter For" field if the DisplayFilterPattern property is True, and the DisplayFilterDate property is False. If the drop down filter window displays "Filter For" field, and user types the filter inside, the FilterType property of the Column is set to exPattern, and Filter property of the Column object specifies the filter being typed.



property Column.DisplaySortIcon as Boolean

Retrieves or sets a value indicating whether the sort icon is visible on column's header, while the column is sorted.

Type	Description
Boolean	A boolean expression indicating whether the sort icon is visible on column's header, while the column is sorted.

Use the DisplaySortIcon property to hide the sort icon. Use the Sort method to sort the items. Use the SortOrder property to sort a column. Use the SingleSort property to allow multiple sort columns. Use the SortOnClick property of control to disable sorting columns by clicking in the column's header.

property Column.Enabled as Boolean

Enables or disables the column.

Type	Description
Boolean	A boolean expression that specifies whether a column is enabled or disabled.

If the Enabled property is False, then all cells of the column are disabled, no matter if the CellEnabled property is true. Use the Enabled property to enable or disable a column. If a cell of radio or check type is disabled, then the cell's state cannot be changed. Use the EnableItem property to disable an item.

property Column.Filter as String

Specifies the column's filter when the filter type is exFilter, exPattern, exDate, exNumeric, exCheck or exImage.

Type	Description
String	A string expression that specifies the column's filter.

- If the FilterType property is **exFilter** the Filter property indicates the list of values being included when filtering. The values are separated by '|' character. For instance if the Filter property is "CellA|CellB" the control includes only the items that have captions like: "CellA" or "CellB".

- If the FilterType is **exPattern** the Filter property defines the list of patterns used in filtering. The list of patterns is separated by the '|' character. A pattern filter may contain the wild card characters like '?' for any single character, '*' for zero or more occurrences of any character, '#' for any digit character. The '|' character separates the options in the pattern. For instance: '1*|2*' specifies all items that start with '1' or '2'.

- If the FilterType property is **exDate**, the Filter property should be of "*[dateFrom] to [dateTo]*" format, and it indicates that only items between a specified range of dates will be included. If the dateFrom value is missing, the control includes only the items before the dateTo date, if the dateTo value is missing, the control includes the items after the dateFrom date. If both dates (dateFrom and dateTo) are present, the control includes the items between this interval of dates. For instance, the "2/13/2004 to" includes all items after 2/13/2004 inclusive, or "2/13/2004 to Feb 14 2005" includes all items between 2/13/2004 and 2/14/2004.

- If the FilterType property is **exNumeric**, the Filter property may include operators like <, <=, =, <>, >= or > and numbers to define rules to include numbers in the control's list. The Filter property should be of the following format "*operator number [operator number ...]*". For instance, the "> 10" indicates all numbers greater than 10. The "<>10 <> 20" filter indicates all numbers except 10 and 20. The "> 10 < 100" filter indicates all numbers greater than 10 and less than 100. The ">= 10 <= 100 <> 50" filter includes all numbers from 10 to 100 excepts 50. The "10" filter includes only 10 in the list. The "=10 =20" includes no items in the list because after control filters only 10 items, the second rule specifies only 20, and so we have no items. The Filter property may include unlimited rules. A rule is composed by an operator and a number. The rules are separated by space characters.

- If the FilterType property is **exCheck** the Filter property may include "0" for unchecked items, and "1" for checked items. The CellState property specifies the state of the

cell's checkbox. If the Filter property is empty, the filter is not applied to the column, when ApplyFilter method is called.

- If the FilterType property is **exImage** the Filter property indicates the list of icons (index of the icon being displayed) being filtered. The values are separated by '|' character. The CellImage property indicates the index of the icon being displayed in the cell. For instance, the '1|2' indicates that the filter includes the cells that display first or the second icon (with the index 1 or 2). The drop down filter window displays the (All) item and the list of icons being displayed in the column.

The Filter property has no effect if the FilterType property is one of the followings: **exAll, exBlanks** and **exNonBlanks**

The ApplyFilter method should be called to update the control's content after changing the Filter or FilterType property. The ClearFilter method clears the Filter and the FilterType properties. Use the FilterCriteria property to specify the filter criteria usinr OR, AND or NOT operators. Use the CustomFilter property to define you custom filters.

property Column.FilterBarDropDownWidth as Double

Specifies the width of the drop down filter window proportionally with the width of the column.

Type	Description
Double	A double expression that indicates the width of the drop down filter window proportionally with the width of the column. If the FilterBarDropDownWidth expression is negative, the absolute value indicates the width of the drop down filter window in pixels. Else, the value indicates how many times the width of the column is multiply to get the width of the drop down filter window.

By default, the FilterBarDropDownWidth property is 1, and so, the width of the drop down filter window coincides with the width of the column. Use the Width property to specify the width of the column. Use FilterBarDropDownHeight property to specify the height of the drop down filter window. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the DisplayFilterButton property to display a filter button to the column's caption. Use the Description property to define predefined strings in the filter bar. Use the ShowFilter method to show programmatically the column's drop down filter window.

The following VB sample specifies that the width of the drop down filter window is double of the column's width:

```
With List1.Columns(0)
   .FilterBarDropDownWidth = 2
End With
```

The following VB sample specifies that the width of the drop down filter window is 150 pixels:

```
With List1.Columns(0)
   .FilterBarDropDownWidth = -150
End With
```

property Column.FilterList as FilterListEnum

Specifies whether the drop down filter list includes visible or all items.

Type	Description
FilterListEnum	A FilterListEnum expression that indicates the items being included in the drop down filter list

By default, the FilterList property is exAllItems. Use the FilterList property to specify the items being included in the column's drop down filter list. Use the DisplayFilterButton property to display the column's filter bar button. The DisplayFilterDate property specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for. Use FilterBarDropDownWidth property to specify the width of the drop down filter window.

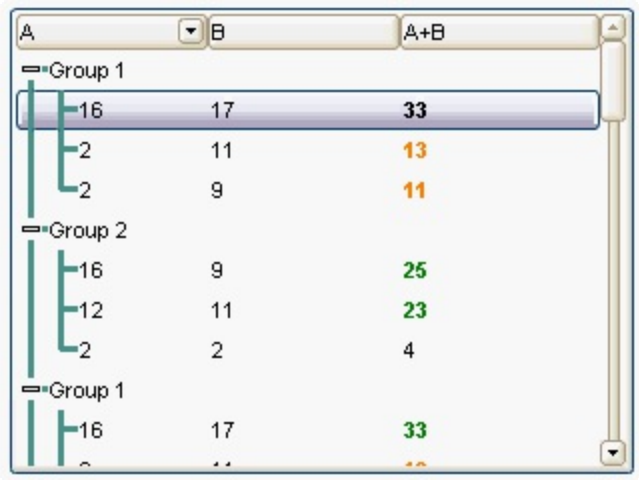
property Column.FilterOnType as Boolean

Filters the column as user types characters in the drop down filter window.

Type	Description
Boolean	A Boolean expression that specifies whether the column gets filtered as the user types characters in the drop down filter window.

By default, the FilterOnType property is False. The Filter-On-Type feature allows you to filter the control's data based on the typed characters. Use the DisplayFilterButton property to add a drop down filter button to the column's header. The Filter-On-Type feature works like follows: User clicks the column's drop down filter button, so the drop down filter window is shown. Use starts type characters, and the control filters the column based on the typed characters as it includes all items that starts with typed characters, if the AutoSearch property is exStartWith, or include in the filter list only the items that contains the typed characters, if the AutoSearch property is exContains. Click the X button on the filterbar, and so the control removes the filter, and so all data is displayed. The control fires the FilterChange event to notify whether the control applies a new filter to control's data. Once, the FilterOnType property is set on True, the column's FilterType property is changed to exPattern, and the the Filter property indicates the typed string. Use the FilterCriteria property to specify the expression being used to filter the control's data when multiple columns are implied in the filter. Use the Description property to customize the text being displayed in the drop down filter window. Use the FilterHeight property to specify the height of the control's filterbar that's displayed on the bottom side of the control, once a filter is applied. The "Filter For" (pattern) field in the drop down filter window is always shown if the FilterOnType property is True, no matter of the DisplayFilterPattern property.

The following screen shot shows how the data gets filtered when the user types characters in the Filter-On-Type columns:



Steps:

- The user clicks the drop down filter window, in the column A
- The "Filter For:" field is shown, and it waits for the user to start type characters.
- As user types characters, the column gets filtered the items.

property Column.FilterType as FilterTypeEnum

Specifies the column's filter type.

Type	Description
FilterTypeEnum	A FilterTypeEnum expression that indicates the filter's type.

The FilterType property defines the filter's type. By default, the FilterType is exAll. No filter is applied if the FilterType is exAll. The Filter property defines the column's filter. Use the DisplayFilterButton property to display the column's filter button. Use the FilterCriteria property to specify the filter criteria usinr OR, AND or NOT operators. Use the CustomFilter property to define you custom filters.

The ApplyFilter method should be called to update the control's content after changing the Filter or FilterType property. The ClearFilter method clears the Filter and the FilterType properties.

property Column.FireFormatColumn as Boolean

Retrieves or sets a value that indicates whether the control fires the FormatColumn event in order to format the caption for each cell in the column.

Type	Description
Boolean	A boolean expression that indicates whether the control fires the FormatColumn event in order to format the caption for each cell in the column.

By default, the FireFormatColumn property is False. If the FireFormatColumn property is True, the control fires the FormatColumn event each time when a cell requires to be displayed. The FormatColumn event lets the user to provide the cell's caption before it is displayed on the control's list. For instance, the FormatColumn event is useful when the column cells contains prices(numbers), and you want to display that column formatted as currency, like $150 instead 150. Also, using the FormatColumn event, you can display the result of some operations within an item, such of totals. *Newer versions of the component provides the FormatColumn property that helps formatting a cell using the several predefined functions without using the control's event FormatColumn.*

Before running any of the following samples, please make sure that the control contains more than 3 columns, and the third column has the FireFormatColumn property on True. The following VB sample displays the sum of the first two cells, and put the result on the third one:

```
Private Sub List1_FormatColumn(ByVal ItemIndex As Long, ByVal ColIndex As Long, Value
As Variant)
On Error Resume Next
   With List1.Items
      Value = Int(.Caption(ItemIndex, 0)) + Int(.Caption(ItemIndex, 1))
   End With
End Sub
```

The following VB sample displays long date format, using the FormatDateTime function:

```
Private Sub List1_FormatColumn(ByVal ItemIndex As Long, ByVal ColIndex As Long, Value
As Variant)
On Error Resume Next
   Value = FormatDateTime(Value, vbLongDate)
End Sub
```

The following C++ sample displays the sum of the first two cells, and put the result on the

third one:

```
void OnFormatColumnList1(long ItemIndex, long ColIndex, VARIANT FAR* Value)
{
   CItems items = m_list.GetItems();
   long newValue = V2I( &items.GetCaption( ItemIndex, COleVariant( long(0) ) ) );
   newValue += V2I( &items.GetCaption( ItemIndex, COleVariant( long(1) ) ) );
   V_VT( Value ) = VT_I4;
   V_I4( Value ) = newValue;
}
```

where the V2I function converts a VARIANT value to a long expression,

```
static long V2I( VARIANT* pv, long nDefault = 0 )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return nDefault;

      COleVariant vt;
      vt.ChangeType( VT_I4, pv );
      return V_I4( &vt );
   }
   return nDefault;
}
```

The following C++ sample displays long date format:

```
void OnFormatColumnList1(long ItemIndex, long ColIndex, VARIANT FAR* Value)
{
   COleDateTime date( *Value );
   COleVariant vtNewValue( date.Format( _T("%A, %B %d, %Y") ) );
   VariantCopy( Value, vtNewValue );
}
```

The following VB.NET sample displays the sum of the first two cells, and put the result on the third one:

```
Private Sub AxList1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_FormatColumnEvent) Handles AxList1.FormatColumn
   With AxList1.Items
      Dim newValue As Integer = Integer.Parse(.Caption(e.itemIndex, 0),
Globalization.NumberStyles.Any)
      newValue = newValue + Integer.Parse(.Caption(e.itemIndex, 1),
Globalization.NumberStyles.Any)
      e.value = newValue
   End With
End Sub
```

The following VB.NET sample displays long date format:

```
Private Sub AxList1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_FormatColumnEvent) Handles AxList1.FormatColumn
   e.value = DateTime.Parse(e.value).ToLongDateString()
End Sub
```

The following C# sample displays the sum of the first two cells, and put the result on the third one:

```
private void axList1_FormatColumn(object sender,
AxEXLISTLib._IListEvents_FormatColumnEvent e)
{
   int newValue = int.Parse(axList1.Items.get_Caption(e.itemIndex, 0).ToString());
   newValue += int.Parse(axList1.Items.get_Caption(e.itemIndex, 1).ToString());
   e.value = newValue;
}
```

The following C# sample displays long date format:

```
private void axList1_FormatColumn(object sender,
AxEXLISTLib._IListEvents_FormatColumnEvent e)
{
   e.value = DateTime.Parse(e.value.ToString()).ToLongDateString();
}
```

The following VFP sample displays the sum of the first two cells, and put the result on the third one:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex, value

with thisform.List1.Items
   value = .Caption(itemindex,0) + .Caption(itemindex,1)
endwith
```

property Column.FormatColumn as String

Specifies the format to display the cells in the column.

Type	Description
String	A string expression that defines the format to display the cell, including HTML formatting, if the cell supports it.

By default, the FormatColumn property is empty. The cells in the column use the provided format only if is valid (not empty, and syntactically correct), to display data in the column. The FormatColumn property provides a format to display all cells in the column using a predefined format. The expression may be a combination of variables, constants, strings, dates and operators, and value. The *value* operator gives the value to be formatted. A string is delimited by ", ` or ' characters, and inside they can have the starting character preceded by \ character, ie "\"This is a quote\"". A date is delimited by # character, ie #1/31/2001 10:00# means the January 31th, 2001, 10:00 AM. The cell's HTML format is applied only if the CaptionFormat or Def(exCaptionFormat) is exHTML. If valid, the FormatColumn is applied to all cells for which the CellCaptionFormat property is not exComputedField. This way you can specify which cells use or not the FormatColumn property. The ComputedField property indicates the formula of the computed column.

For instance:

- the "*currency(value)*" displays the column using the current format for the currency ie, 1000 gets displayed as $1,000.00
- the "*longdate(date(value))*" converts the value to a date and gets the long format to display the date in the column, ie #1/1/2001# displays instead Monday, January 01, 2001
- the "*'' + ((0:=proper(value)) left 1) + '' + (=:0 mid 2)*" converts the name to proper, so the first letter is capitalized, bolds the first character, and let unchanged the rest, ie a "mihai filimon" gets displayed "**M**ihai Filimon".
- the "*len(value) ? ((0:=dbl(value)) < 10 ? '<fgcolor=808080><font ;7>' : '') + currency(=:0)*" displays the cells that contains not empty daya, the value in currency format, with a different font and color for values less than 10, and bolded for those that are greater than 10, as can see in the following screen shot in the column (A+B+C):

Name	A	B	C	A+B+C
Item 1	~~7 +~~	3 +	1 =	**$11.00**
Item 2	2 +	~~5 +~~	~~12 =~~	**$19.00**
Item 3	2 +	2 +	4 =	$8.00
Item 4	2 +	~~9 +~~	4 =	**$15.00**

The **value** keyword in the FormatColumn/FormatCell property indicates the value to be formatted.

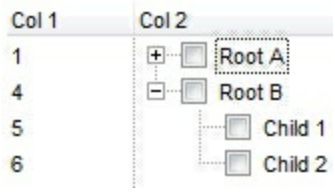
The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The Caption property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*

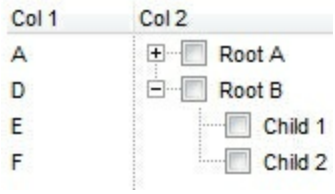
Other known operators for auto-numbering are:

- number **index** 'format', indicates the index of the item. The first added item has the index 0, the second added item has the index 1, and so on. The index of the item remains the same even if the order of the items is changed by sorting. For instance, 1 index '' gets the index of the item starting from 1 while 100 index '' gets the index of the item starting from 100. The number indicates the starting index, while the format is a set of characters to be used for specifying the index. If the format is missing, the index of the item is formatted as numbers. For instance: 1 index 'A-Z' gets the index as A, B, C... Z, BA, BB, ... BZ, CA, ... The 1 index 'abc' gives the index as: a,b,c,ba,bb,bc,ca,cb,cc,.... You can use other number formatting function to format the returned value. For instance "1 index '' format '0||2|:'" gets the numbers grouped by 2 digits and separated by : character.

  In the following screen shot the FormatColumn("Col 1") = "1 index ""

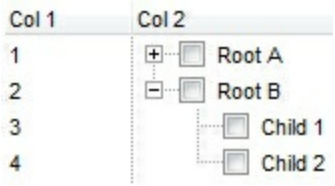


  In the following screen shot the FormatColumn("Col 1") = "1 index 'A-Z'"

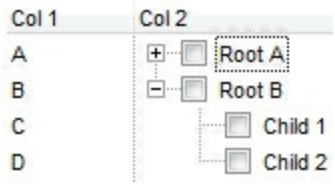


- number **apos** 'format' indicates the absolute position of the item. The first displayed item has the absolute position 0 (scrolling position on top), the next visible item is 1, and so on. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance, 1 apos '' gets the absolute position of the item starting from 1, while 100 apos '' gets the position of the item starting from 100. If the format is missing, the absolute position of the item is formatted as numbers.

In the following screen shot the FormatColumn("Col 1") = "1 apos ""

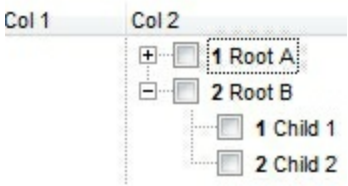


In the following screen shot the FormatColumn("Col 1") = "1 apos 'A-Z'"

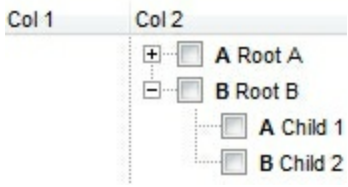


- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100. If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied, the opos gets the position of the item in the list of unfiltered items, while the pos gets the position of the item in the filtered list.*

  In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos '' + ' ' + value"



  In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos 'A-Z' + ' ' + value"

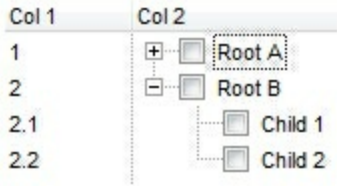


- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for

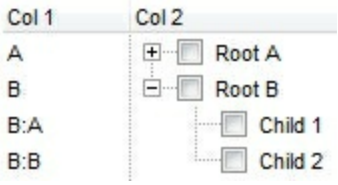
specifying the position.For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100. If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied, the opos gets the position of the item in the list of unfiltered items, while the pos gets the position of the item in the filtered list.*

- number **rpos** 'format' indicates the relative recursive position of the item. The recursive position indicates the position of the parent items too. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is of the following type "delimiter|format|format|...". If the format is missing, the delimiter is . character, and the positions are formatted as numbers. The format is applied consecutively to each parent item, from root to item itself.

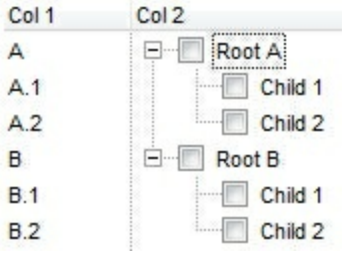
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ""



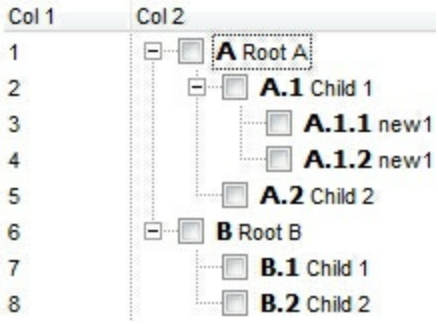
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"



  In the following screen shot the FormatColumn("Col 1") = "1 rpos '.|A-Z|'"



  In the following screen shot the FormatColumn("Col 1") = "1 apos "" and FormatColumn("Col 2") = "'' + 1 rpos '.|A-Z|' + ' ' + value"



- number **rindex** 'format', number **rapos** 'format' and number **ropos** 'format' are working similar with number **rpos** 'format', excepts that they gives the index, absolute position, or the old child position.

This property/method supports predefined constants and operators/functions as described here.

The following **VB** sample shows how can I display the column using currency:

```
With List1
   .Columns.Add("Currency").FormatColumn = "currency(dbl(value))"
   With .Items
      .Add "1.23"
      .Add "2.34"
      .Add "0"
      .Add 5
      .Add "10000.99"
   End With
End With
```

The following **VB.NET** sample shows how can I display the column using currency:

```
With AxList1
   .Columns.Add("Currency").FormatColumn = "currency(dbl(value))"
   With .Items
      .Add "1.23"
      .Add "2.34"
      .Add "0"
      .Add 5
      .Add "10000.99"
   End With
End With
```

The following **C++** sample shows how can I display the column using currency:

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

   #import "C:\\Windows\\System32\\ExList.dll"
   using namespace EXLISTLib;
*/
EXLISTLib::IListPtr spList1 = GetDlgItem(IDC_LIST1)->GetControlUnknown();
((EXLISTLib::IColumnPtr)(spList1->GetColumns()->Add(L"Currency")))-
>PutFormatColumn(L"currency(dbl(value))");
EXLISTLib::IItemsPtr var_Items = spList1->GetItems();
   var_Items->Add("1.23");
   var_Items->Add("2.34");
   var_Items->Add("0");
   var_Items->Add(long(5));
   var_Items->Add("10000.99");
```

The following **C#** sample shows how can I display the column using currency:

```
(axList1.Columns.Add("Currency") as EXLISTLib.Column).FormatColumn =
"currency(dbl(value))";
EXLISTLib.Items var_Items = axList1.Items;
   var_Items.Add("1.23");
   var_Items.Add("2.34");
   var_Items.Add("0");
   var_Items.Add(5);
   var_Items.Add("10000.99");
```

The following **VFP** sample shows how can I display the column using currency:

```
with thisform.List1
   .Columns.Add("Currency").FormatColumn = "currency(dbl(value))"
   with .Items
      .Add("1.23")
      .Add("2.34")
      .Add("0")
```

```
        .Add(5)
        .Add("10000.99")
    endwith
endwith
```

property Column.HeaderAlignment as AlignmentEnum

Specifies the alignment of the column's caption.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the caption in the column's header.

Use the HeaderAlignment property to align the column's caption inside the column's header. Use the Alignment property to align the cells into a column. Use the HeaderImageAlignment property to align the column's icon inside the column's header. Use the CellHAlignment property to align a cell.

property Column.HeaderBold as Boolean

Retrieves or sets a value that indicates whether the column's caption should appear in bold.

Type	Description
Boolean	A boolean expression that indicates whether the column's caption should appear in bold.

The HeaderBold property specifies whether the column's caption should appear in bold. Use the CellBold or ItemBold properties to specify whether the cell or item should appear in bold. Use the HTMLCaption property to specify portions of the caption using different colors, fonts. Use the HeaderItalic, HeaderUnderline or HeaderStrikeOut property to specify different font attributes when displaying the column's caption.

property Column.HeaderImage as Long

Retrieves or sets a value indicating the index of an Image in the Images collection, that is displayed in the column's header.

Type	Description
Long	A long value indicating the index of an Image in the Images collection, that is displayed in the column's header. The last 7 bits in the high significant byte of the long 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.

Use the HeaderImage property to add an icon to the column's caption. Use the HeaderImageAlignment property to change the column header image's alignment. Use the CellImage to assign a single icon to a cell. Use the CellImages to assign multiple icons to a cell. Use the CellPicture to attach a picture object to a cell.

property Column.HeaderImageAlignment as AlignmentEnum

Retrieves or sets the alignment of the image into the column's header.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the image into the column's header.

By default, the image is left aligned. Use the HeaderImageAlignment property to aligns the icon in the column's header. Use the HeaderImage property to attach an icon to the column's header.

property Column.HeaderItalic as Boolean

Retrieves or sets a value that indicates whether the column's caption should appear in italic.

Type	Description
Boolean	A boolean expression that indicates whether the column's caption should appear in italic.

Use the HeaderItalic property to specify whether the column's caption should appear in italic. Use the CellItalic or ItemItalic properties to specify whether the the cell or the item should appear in italic. Use the HeaderBold, HeaderUnderline or HeaderStrikeOut property to specify different font attributes when displaying the column's caption.

property Column.HeaderStrikeOut as Boolean

Retrieves or sets a value that indicates whether the column's caption should appear in strikeout.

Type	Description
Boolean	A boolean expression that indicates whether the column's caption should appear in strikeout.

Use the HeaderStrikeOut property to specify whether the column's caption should appear in strikeout. Use the CellStrikeOut or ItemStrikeOut properties to specify whether the cell or the item should appear in strikeout. Use the HeaderItalic, HeaderUnderline or HeaderBold property to specify different font attributes when displaying the column's caption.

property Column.HeaderUnderline as Boolean

Retrieves or sets a value that indicates whether the column's caption should appear in underline.

Type	Description
Boolean	A boolean expression that indicates whether the column's caption should appear in underline.

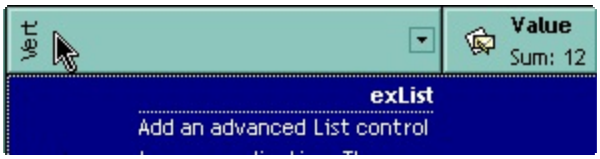
Use the HeaderUnderline property to specify whether the column's caption should appear in underline. Use the CellUnderline or ItemUnderline properties to specify whether the cell or the item should appear in underline. Use the HeaderItalic, HeaderBold or HeaderStrikeOut property to specify different font attributes when displaying the column's caption.

property Column.HeaderVertical as Boolean

Specifies whether the column's header is vertically displayed.

Type	Description
Boolean	A boolean expression that indicates whether the column's caption is vertically printed.

Use the HeaderVertical property to display vertically the column's caption. Use the Caption property to specify the column's caption. Use the HTMLCaption property to specify the column's caption using built-in HTML format. Use the HeaderAlignment property to align the column's caption.



property Column.HTMLCaption as String

Retrieves or sets the text in HTML format displayed in the column's header.

Type	Description
String	A string expression that indicates the column's caption using built-in HTML tags.

If the HTMLCaption property is empty, the Caption property is displayed in the column's header. If the HTMLCaption property is not empty, the control uses it when displaying the column's header. Use the HeaderHeight property to change the height of the control's header bar. The list of built-in HTML tags supported are here.

property Column.Index as Long

Returns a value that represents the index of an object in a collection.

Type	Description
Long	A long expression that indicates the column's index.

The Index property is a read-only property. The control assigns the index of the column when adding new columns. Use the Position property to change the column's position. The Columns collection is zero based, so the Index property starts at 0. The last added column has the Index set to Columns.Count - 1. When a column is removed from the collection, the control updates all indexes. Use the Visible property to hide a column. Use the Columns property to access column from it's index.

property Column.Key as String

Retrieves or sets a the column's key.

Type	Description
String	A string expression that defines the column's key

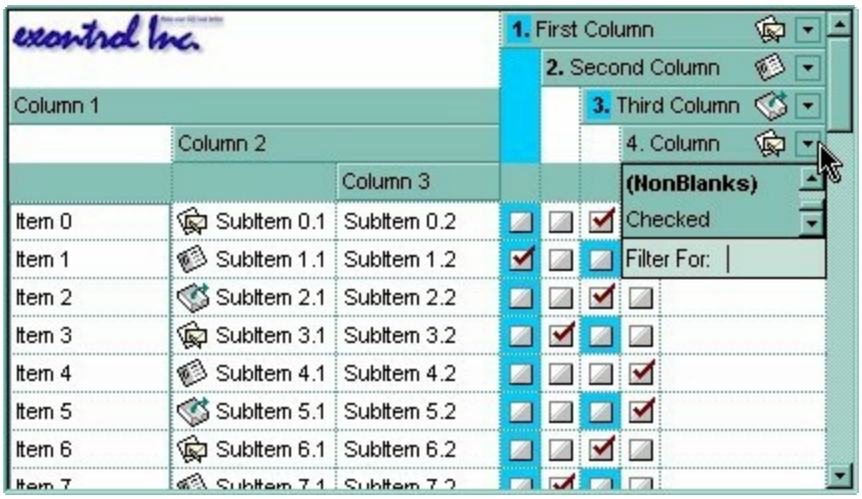
The column's key defines a column when using the Item property. Use the Index or the Key property to identify a column, when using the Columns property.

property Column.LevelKey as Variant

Retrieves or sets a value that indicates the key of the column's level.

Type	Description
Variant	A Variant expression that indicates the key of the column's level.

By default, the LevelKey is empty. The control's header displays multiple levels if there are two or more neighbor columns with the same non empty level key. The HeaderHeight property specifies the height of one level when multiple levels header is on. Use the BackColorLevelHeader property to specify the control's level header area. Use the PictureLevelHeader property to assign a picture on the control's header. The BackColorHeader property specifies the background color for column's captions.



property Column.MaxWidthAutoResize as Long

Retrieves or sets a value that indicates the maximum column's width when the WidthAutoResize is True.

Type	Description
Long	A long expression that the maximum column's width when the WidthAutoResize is True.

If the WidthAutoResize property is False, the MaxWidthAutoResize and MinWidthAutoResize properties have no effect. The MaxWidthAutoResize property specifies the maximum column's width. The control recalculates the column's width each time when an item is expanded or collapsed. If the MaxWidthAutoResize property is -1, there is no maximum value for the column's width. Use the WidthAutoResize, MaxWidthAutoResize and MinWidthAutoResize properties when you don't want to have truncated the caption for cells in the column. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area.

property Column.MinWidthAutoResize as Long

Retrieves or sets a value that indicates the minimum column width when the WidthAutoResize is True.

Type	Description
Long	A long expression that indicates the minimum column's width when the WidthAutoResize is True.

If the WidthAutoResize property is False, the MaxWidthAutoResize and MinWidthAutoResize properties have no effect. The MinWidthAutoResize property specifies the minimum column's width. The control recalculates the column's width each time when an item is expanded or collapsed. Use the WidthAutoResize, MaxWidthAutoResize and MinWidthAutoResize properties when you don't want to have truncated the caption for cells in the column. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area.

property Column.Position as Long

Retrieves or sets a value that indicates the position of the column in the header bar area.

Type	Description
Long	A long expression that indicates the position of the column in the header bar area

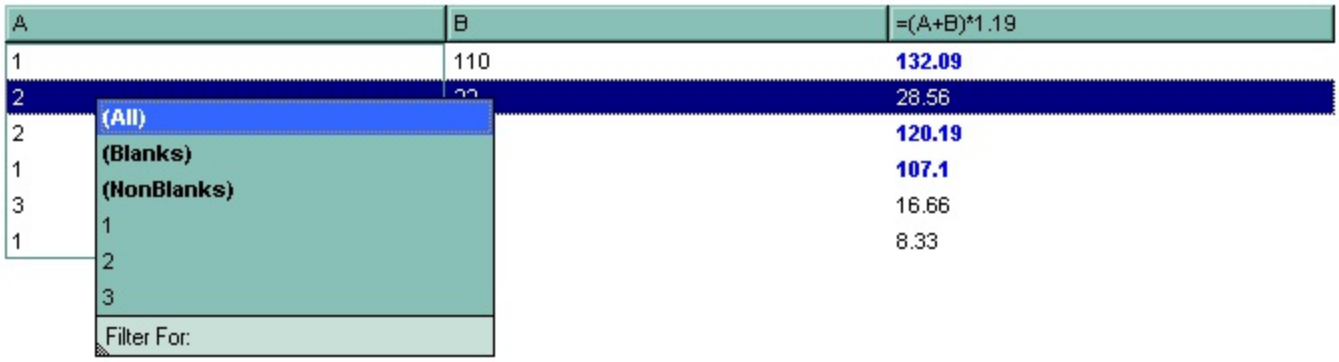
The column's index is not the same with the column's position. The Index property of Column cannot be changed by the user. Use the Position property to change the column's position. Use the SortPosition property to change the position of the column in the control's sort bar. Use the Visible property to hide a column. Use the Width property to specify the column's width.

method Column.ShowFilter ([Options as Variant])

Shows the column's filter window.

Type	Description
Options as Variant	A string expression that indicates the position (in screen coordinates) and the size (in pixels) where the drop down filter window is shown. The Options parameter is composed like follows: • the first parameter indicates the X coordinate in screen coordinate, -1 if the current cursor position is used, or empty if the coordinate is ignored • the second parameter indicates the Y coordinate in screen coordinate, -1 if the current cursor position is used, or empty if the coordinate is ignored • the third parameter indicates the width in pixels of the drop down window, or empty if the width is ignored • the forth parameter indicates the height in pixels of the drop down window, or empty if the height is ignored By default, the drop down filter window is shown at its default position bellow the column's header.

Use the ShowFilter method to show the column's drop down filter programmatically. By default, the drop down filter window is shown only if the user clicks the filter button in the column's header, if the DisplayFilterButton property is True. The drop down filter window if the user selects a predefined filter, or enters a pattern to match. If the Options parameter is missing, or all parameters inside the Options are missing, the size of the drop down filter window is automatcially computed based on the FilterBarDropDownWidth property and FilterBarDropDownHeight property. Use the ColumnFromPoint property to get the index of the column from the point.



For instance, the following VB sample displays the column's drop down filter window when the user right clicks the control:

```
Private Sub List1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   If (Button = 2) Then
      With List1.Columns
         With .Item(List1.ColumnFromPoint(-1, 0))
            .ShowFilter "-1,-1,200,200"
         End With
      End With
   End If
End Sub
```

The following VB.NET sample displays the column's drop down filter window when the user right clicks the control:

```
Private Sub AxList1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseUpEvent) Handles AxList1.MouseUpEvent
   If (e.button = 2) Then
      With AxList1.Columns
         With .Item(AxList1.get_ColumnFromPoint(-1, 0))
            .ShowFilter("-1,-1,200,200")
         End With
      End With
   End If
End Sub
```

The following C# sample displays the column's drop down filter window when the user right clicks the control:

```
private void axList1_MouseUpEvent(object sender,
AxEXLISTLib._IListEvents_MouseUpEvent e)
{
   if (e.button == 2)
   {
      EXLISTLib.Column c = axList1.Columns[axList1.get_ColumnFromPoint(-1, 0)];
      c.ShowFilter("-1,-1,200,200");
   }
}
```

The following C++ sample displays the column's drop down filter window when the user right clicks the control:

```
void OnMouseUpList1(short Button, short Shift, long X, long Y)
{
    m_list.GetColumns().GetItem( COleVariant( m_list.GetColumnFromPoint( -1, 0 ) )
).ShowFilter( COleVariant( "-1,-1,200,200" ) );
}
```

The following VFP sample displays the column's drop down filter window when the user right clicks the control:

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

if ( button = 2 ) then
    With thisform.List1.Columns
        With .Item(thisform.List1.ColumnFromPoint(-1, 0))
            .ShowFilter("-1,-1,200,200")
        EndWith
    EndWith
endif
```

property Column.SortOrder as SortOrderEnum

Specifies the column's sort order.

Type	Description
SortOrderEnum	A SortOrderEnum expression that indicates the column's sort order.

The SortOrder property determines the column's sort order. By default, the SortOrder property is SortNone. Use the SortOrder property to sort a column at runtime. Use the SortType property to determine the way how the column is sorted. Use the AllowSort property to avoid sorting a column when the user clicks the column. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. If the control supports sorting by multiple columns, the SortOrder property adds or removes the column to the sorting columns collection. For instance, if the SortOrder property is set to SortAscending or SortDescending the column is added to the sorting columns collection. If the SortOrder property is set to SortNone the control removes the column from its sorting columns collection. The Sort event is fired when the user sorts a column. The SortPosition property changes the position of the column in the control's sort bar. Use the DefaultSortOrder property to specify the column's default sort order, when the user first clicks the column's header. Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items as they are listed.

- Using the SortOrder property of the Column object. The SortOrder property displays the sorting icon in the column's header if the DisplaySortIcon property is True.

```
List1.Columns(ColIndex).SortOrder = SortAscending
```

- Using the Sort method of Items object. The following sample sort descending the list of root items on the "Column 2"

```
List1.Items.Sort "Column 2", False
```

property Column.SortPosition as Long

Returns or sets a value that indicates the position of the column in the sorting columns collection.

Type	Description
Long	A long expression that indicates the position of the column in the control's sort bar. The collection is 0 - based.

Use the SortPosition to change programmatically the position of the column in the control's sort bar. Use the SingleSort property to allow sorting by multiple columns. Use the SortBarVisible property to show the control's sort bar. Use the SortOrder property to add columns to the control's sort bar. The control fires the Sort event when the user sorts a column. Use the ItemBySortPosition property to get the columns being sorted in their order. Use the AllowSort property to avoid sorting a column when the user clicks the column.

property Column.SortType as SortTypeEnum

Returns or sets a value that indicates the way the control sorts the values for a column.

Type	Description
SortTypeEnum	A SortTypeEnum expression that indicates the way how control sorts the column.

The SortType property specifies the way how a column is sorted. By default, the column's SortType is String. Use the SortType property to specifies how the control will sort the column. Use the Sort method to sort a column. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. The SortOrder property determines the column's sort order. The Sort event is fired when the user sorts a column. The SortPosition property changes the position of the column in the sorting columns collection.

property Column.ToolTip as String

Specifes the column's tooltip description.

Type	Description
String	A string expression that defines the column's tooltip. The column's tooltip supports built-in HTML format.

By default, the Tooltip property is empty. Use the ToolTip property to assign a tooltip to a column. The column's tooltip shows up when the cursor is over the header of the column. Use the CellToolTip property to assign a tooltip to a cell. The control fires the ToolTip event when the column's tooltip is about to be displayed. The ToolTipWidth property specifies a value that indicates the width of the tooltip window, in pixels.

property Column.Visible as Boolean

Retrieves or sets a value indicating whether the column is visible or hidden.

Type	Description
Boolean	A boolean expression indicating whether the column is visible or hidden.

Use the Visible property to hide a column. Use the Width property to resize the column. The ColumnAutoResize property specifies whether the visible columns fit the control's client area. Use the Position property to specify the column's position. Use the HeaderVisible property to show or hide the control's header bar. Use the ColumnFromPoint property to get the column from point. Use the Remove method to remove a column.

property Column.Width as Long

Retrieves or sets the column's width.

Type	Description
Long	A long expression that indicates the column's width.

The Width property specifies the column's width in pixels. Use the AutoWidth property to compute the required width to fit the entire column. Use the WidthAutoResize property to automatically resize the column while the user expands or collapses items. Use the Visible property to hide a column. Use the SortBarColumnWidth property to specify the column's head in the control's sort bar. Use the ColumnAutoResize property to fit all visible columns in the control's client area. Use FilterBarDropDownWidth property to specify the width of the drop down filter window.

property Column.WidthAutoResize as Boolean

Retrieves or sets a value that indicates whether the column is automatically resized according to the width of the contents within the column.

Type	Description
Boolean	A boolean expression that indicates whether the column is automatically resized according to the width of the contents within the column.

If the WidthAutoResize property is True, the column's width is resized after user edits the cell. Also, the column's width is refreshed if the user adds new items to the control. If the WidthAutoResize property is True, the column's width is not larger than MaxWidthAutoResize value, and it is not less than MinWidthAutoResize value. You can use the AutoWidth property to computes the column's width to fit its content. For instance, if you have a tree with one column, and this property True, you can simulate a simple tree, because the control will automatically add a horizontal scroll bar when required. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area. If the WidthAutoResize property is True, the user is not able to resize the column, so the AllowSizing property has no effect in this case.

Columns object

The Columns object holds a collection of Column objects. The Columns collection supports the following properties and methods:

Name	Description
Add	Adds a Column object to the collection 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 Column of the Columns collection.
ItemBySortPosition	Returns a Column object giving its sorting position.
Remove	Removes a specific member from the Columns collection.

method Columns.Add (ColumnCaption as String)

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

Type	Description
ColumnCaption as String	A string expression that defines the column's caption

Return	Description
Variant	A Column object that represents the newly created column.

By default, the control has no columns. Use Add method to add new columns to the control. If the control contains no columns, adding new items fail. Use the Remove method to remove a specific column. The control fires the AddColumn event when a new column is added. The DataSource property automatically adds new columns for each field found in the recordset, and add new items for each record in the recordset. You can use Add method to add computed columns. Use the HTLMCaption property to display the column's caption using HTML tags. Use the Visible property to hide a column. Use the BeginUpdate and EndUpdate methods to prevent control from painting while adding columns and items. Use the Add or PutItems method to add new items to the control. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample adds columns from a record set:

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Orders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Opens the table using static mode
With List1
   .BeginUpdate
   .ColumnAutoResize = False
   With .Columns
      For Each f In rs.Fields
         .Add f.Name
      Next
   End With
   .PutItems rs.getRows()
   .EndUpdate
End With
```

The following VC sample adds a column:

```
#include "Columns.h"
#include "Column.h"
CColumns columns = m_list.GetColumns();
CColumn column( V_DISPATCH( &columns.Add( "Column 1" ) ) );
column.SetHeaderBold( TRUE );
```

The following VB.NET sample adds a column:

```
With AxList1.Columns
   With .Add("Column 1")
      .HeaderBold = True
   End With
End With
```

The Add method returns a Column object in a VARIANT value, so you can use a code like follows:

```
With AxList1.Columns
   Dim c As EXLISTLib.Column
   c = .Add("Column 1")
   With c
      .HeaderBold = True
   End With
End With
```

this way, you can have the properties of the column at design time when typing the '.' character.

The following C# sample adds a column:

```
EXLISTLib.Column column = axList1.Columns.Add( "Column 1" ) as EXLISTLib.Column;
column.HeaderBold = true;
```

The following VFP sample adds a column:

```
with thisform.List1.Columns.Add( "Column 1" )
   .HeaderBold = .t.
endwith
```

method Columns.Clear ()

Removes all objects in a collection.

Type	Description

Use the Clear method to remove all columns in the Columns collection. If the Clear method is called, the control removes also all items. Use the Remove method to Remove a particular column. The Clear method calls RemoveColumn event for each column deleted. Use the RemoveAll method to remove all items in the control.

property Columns.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	Counts the columns in the collection.

The Count property counts the columns in the collection. Use the Columns property to access the control's Columns collection. Use the Item property to access a column by its index or key. Use the Add method to add new columns to the control. Use the Remove method to remove a column. Use the Clear method to clear the columns collection.

The following VB sample enumerates the columns in the control:

```
For Each c In Tree1.Columns
   Debug.Print c.Caption
Next
```

The following VB sample enumerates the columns in the control:

```
For i = 0 To Tree1.Columns.Count - 1
   Debug.Print Tree1.Columns(i).Caption
Next
```

The following VC sample enumerates the columns in the control:

```
#include "Columns.h"
#include "Column.h"
CColumns columns = m_tree.GetColumns();
for ( long i = 0; i < columns.GetCount(); i++ )
{
   CColumn column = columns.GetItem( COleVariant( i ) );
   OutputDebugString( column.GetCaption() );
}
```

The following VB.NET sample enumerates the columns in the control:

```
With AxTree1.Columns
   Dim i As Integer
   For i = 0 To .Count - 1
      Debug.WriteLine(.Item(i).Caption)
```

```
    Next
End With
```

The following C# sample enumerates the columns in the control:

```
EXTREELib.Columns columns =axTree1.Columns;
for ( int i = 0; i < columns.Count - 1; i++ )
{
    EXTREELib.Column column = columns[i];
    System.Diagnostics.Debug.WriteLine( column.Caption );
}
```

The following VFP sample enumerates the columns in the control:

```
with thisform.Tree1.Columns
    for i = 0 to .Count - 1
        wait window nowait .Item(i).Caption
    next
endwith
```

property Columns.Item (Index as Variant) as Column

Returns a specific Column of the Columns collection.

Type	Description
Index as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Column	A Column object being accessed.

Use the Item property to access to a specific column. The Count property counts the columns in the control. Use the Columns property to access the control's Columns collection.

The following VB sample enumerates the columns in the control:

```
For Each c In Tree1.Columns
   Debug.Print c.Caption
Next
```

The following VB sample enumerates the columns in the control:

```
For i = 0 To Tree1.Columns.Count - 1
   Debug.Print Tree1.Columns(i).Caption
Next
```

The following VC sample enumerates the columns in the control:

```
#include "Columns.h"
#include "Column.h"
CColumns columns = m_tree.GetColumns();
for ( long i = 0; i < columns.GetCount(); i++ )
{
   CColumn column = columns.GetItem( COleVariant( i ) );
   OutputDebugString( column.GetCaption() );
}
```

The following VB.NET sample enumerates the columns in the control:

```
With AxTree1.Columns
   Dim i As Integer
```

```
    For i = 0 To .Count - 1
        Debug.WriteLine(.Item(i).Caption)
    Next
End With
```

The following C# sample enumerates the columns in the control:

```
EXTREELib.Columns columns =axTree1.Columns;
for ( int i = 0; i < columns.Count - 1; i++ )
{
    EXTREELib.Column column = columns[i];
    System.Diagnostics.Debug.WriteLine( column.Caption );
}
```

The following VFP sample enumerates the columns in the control:

```
with thisform.Tree1.Columns
    for i = 0 to .Count - 1
        wait window nowait .Item(i).Caption
    next
endwith
```

property Columns.ItemBySortPosition (Position as Variant) as Column

Returns a Column object giving its sorting position.

Type	Description
Position as Variant	A long expression that indicates the position of column being requested.
Column	A Column object being accessed.

Use the ItemBySortPosition property to get the list of sorted columns in their order. Use the SortPosition property to specify the position of the column in the sorting columns collection. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. Use the SortOrder property to sort a column programmatically. The control fires the Sort event when the user sorts a column.

The following VB sample displays the list of columns being sorted:

```
Dim s As String, i As Long, c As Column
i = 0
With List1.Columns
   Set c = .ItemBySortPosition(i)
   While (Not c Is Nothing)
      s = s + """" & c.Caption & """ " & IIf(c.SortOrder = SortAscending, "A", "D") & " "
      i = i + 1
      Set c = .ItemBySortPosition(i)
   Wend
End With
s = "Sort: " & s
Debug.Print s
```

The following VC sample displays the list of columns being sorted:

```
CString strOutput;
CColumns columns = m_list.GetColumns();
long i = 0;
CColumn column = columns.GetItemBySortPosition( COleVariant( i ) );
while ( column.m_lpDispatch )
{
   strOutput += "\"" + column.GetCaption() + "\" " + ( column.GetSortOrder() == 1 ? "A" :
"D" ) + " ";
```

```
    i++;
    column = columns.GetItemBySortPosition( COleVariant( i ) );
}
strOutput += "\r\n";
OutputDebugString( strOutput );
```

The following C# sample displays the list of columns being sorted:

```
string strOutput = "";
int i = 0;
EXLISTLib.Column column = axList1.Columns.get_ItemBySortPosition( i );
while ( column != null )
{
    strOutput += column.Caption + " " + ( column.SortOrder ==
EXLISTLib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
    column = axList1.Columns.get_ItemBySortPosition( ++i );
}
Debug.WriteLine( strOutput );
```

The following VB.NET sample displays the list of columns being sorted:

```
With AxList1
    Dim s As String, i As Integer, c As EXLISTLib.Column
    i = 0
    With AxList1.Columns
        c = .ItemBySortPosition(i)
        While (Not c Is Nothing)
            s = s + """" & c.Caption & """ " & IIf(c.SortOrder =
EXLISTLib.SortOrderEnum.SortAscending, "A", "D") & " "
            i = i + 1
            c = .ItemBySortPosition(i)
        End While
    End With
    s = "Sort: " & s
    Debug.WriteLine(s)
End With
```

The following VFP sample displays the list of columns being sorted:

```
local s, i, c
i = 0
s = ""
With thisform.List1.Columns
   c = .ItemBySortPosition(i)
   do While (!isnull(c))
      with c
         s = s + "'" + .Caption
         s = s + "' " + Iif(.SortOrder = 1, "A", "D") + " "
         i = i + 1
      endwith
      c = .ItemBySortPosition(i)
   enddo
endwith
s = "Sort: " + s
wait window nowait s
```

method Columns.Remove (Index as Variant)

Removes a specific member from the Columns collection.

Type	Description
Index as Variant	A long expression that indicates the column's index being removed, or a string expression that indicates the column's caption or column's key

The Remove method removes a specific column in the Columns collection. Use Clear method to remove all Column objects. The RemoveColumn event is fired when a column is about to be removed. Use the Visible property to hide a column.

ConditionalFormat object

The conditional formatting feature allows you to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. Use the Add method to add new ConditionalFormat objects. Use the Item property to access a ConditionalFormat object. The ConditionalFormat object supports the following properties and method:

Name	Description
ApplyTo	Specifies whether the format is applied to items or columns.
BackColor	Retrieves or sets the background color for objects that match the condition.
Bold	Bolds the objects that match the condition.
ClearBackColor	Clears the background color.
ClearForeColor	Clears the foreground color.
Enabled	Specifies whether the condition is enabled or disabled.
Expression	Indicates the expression being used in the conditional format.
Font	Retrieves or sets the font for objects that match the criteria.
ForeColor	Retrieves or sets the foreground color for objects that match the condition.
Italic	Specifies whether the objects that match the condition should appear in italic.
Key	Checks whether the expression is syntactically correct.
StrikeOut	Specifies whether the objects that match the condition should appear in strikeout.
Underline	Underlines the objects that match the condition.
Valid	Checks whether the expression is syntactically correct.

property ConditionalFormat.ApplyTo as FormatApplyToEnum

Specifies whether the format is applied to items or columns.

Type	Description
FormatApplyToEnum	A FormatApplyToEnum expression that indicates whether the format is applied to items or to columns. If the ApplyTo property is less than zero, the format is applied to the items.

By default, the format is applied to items. The ApplyTo property specifies whether the format is applied to the items or to the columns. If the ApplyTo property is greater or equal than zero the format is applied to the column with the index ApplyTo. For instance, if the ApplyTo property is 0, the format is applied to the cells in the first column. If the ApplyTo property is 1, the format is applied to the cells in the second column, if the ApplyTo property is 2, the format is applied to the cells in the third column, and so on. If the ApplyTo property is -1, the format is applied to items.

The following VB sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With List1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C++ sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
COleVariant vtEmpty;
CConditionalFormat cf = m_list.GetConditionalFormats().Add( "%1+%2<%0", vtEmpty );
cf.SetBold( TRUE );
cf.SetApplyTo( 1 );
```

The following VB.NET sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With AxList1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C# sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
EXLISTLib.ConditionalFormat cf = axList1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXLISTLib.FormatApplyToEnum)1;
```

The following VFP sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
with thisform.List1.ConditionalFormats.Add("%1+%2<%0")
   .Bold = .t.
   .ApplyTo = 1
endwith
```

property ConditionalFormat.BackColor as Color

Retrieves or sets the background color for objects that match the condition.

Type	Description
Color	A color expression that indicates the background color for the object that match the criteria. 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.

Use the BackColor property to change the background color for items or cells in the column when a certain condition is met. Use the ForeColor property to specify the foreground color for objects that match the criteria. Use the ClearBackColor method to remove the background color being set using previously the BackColor property. If the BackColor property is not set, it retrieves 0. The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column.

property ConditionalFormat.Bold as Boolean

Bolds the objects that match the condition.

Type	Description
Boolean	A boolean expression that indicates whether the objects should appear in bold.

The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column. The following VB sample bolds all cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With List1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C++ sample bolds all cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
COleVariant vtEmpty;
CConditionalFormat cf = m_list.GetConditionalFormats().Add( "%1+%2<%0", vtEmpty );
cf.SetBold( TRUE );
cf.SetApplyTo( 1 );
```

The following VB.NET sample bolds all cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With AxList1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C# sample bolds all cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
EXLISTLib.ConditionalFormat cf = axList1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXLISTLib.FormatApplyToEnum)1;
```

The following VFP sample bolds all cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
with thisform.List1.ConditionalFormats.Add("%1+%2<%0")
   .Bold = .t.
   .ApplyTo = 1
endwith
```

method ConditionalFormat.ClearBackColor ()

Clears the background color.

Type	Description

Use the ClearBackColor method to remove the background color being set using previously the BackColor property. If the BackColor property is not set, it retrieves 0.

method ConditionalFormat.ClearForeColor ()

Clears the foreground color.

Type	Description

Use the ClearBackColor method to remove the foreground color being set using previously the ForeColor property. If the ForeColor property is not set, it retrieves 0.

property ConditionalFormat.Enabled as Boolean

Specifies whether the condition is enabled or disabled.

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

By default, all expressions are enabled. A format is applied only if the expression is valid and enabled. Use the Expression property to specify the format's formula. The Valid property checks whether the formula is valid or not valid. Use the Enabled property to disable applying the format for the moment. Use the Remove method to remove an expression from ConditionalFormats collection.

property ConditionalFormat.Expression as String

Indicates the expression being used in the conditional format.

Type	Description
String	A formal expression that indicates the formula being used in formatting. For instance, "%0+%1>%2", highlights the cells or the items, when the sum between first two columns is greater than the value in the third column

The conditional formatting feature allows you to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. The Expression property specifies a formula that indicates the criteria to format the items or the columns. Use the ApplyTo property to specify when the items or the columns are formatted. Use the Add method to specify the expression at adding time. The Expression property may include variables, constants, operators or ( ) parenthesis. A variable is defined as %n, where n is the index of the column (zero based). For instance, the %0 indicates the first column, the %1, indicates the second column, and so on. The Valid property specifies whether the expression is syntactically correct, and can be evaluated. If the expression contains a variable that is not known, 0 (or empty string) value is used instead. For instance, if your control has 2 columns, and the expression looks like "%2 +%1 ", the %2 does not exist, 0 is used instead. For instance, if the control contains two columns the known variables are %0 and %1.

The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The Caption property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*

This property/method supports predefined constants and operators/functions as described here.

Samples:

1. "**1**", highlights all cells or items. Use this form, when you need to highlight all cells or items in the column or control.
2. "**%0 >= 0**", highlights the cells or items, when the cells in the first column have the value greater or equal with zero
3. "**%0 = 1 and %1 = 0**", highlights the cells or items, when the cells in the first column have the value equal with 0, and the cells in the second column have the value equal with 0
4. "**%0+%1>%2**", highlights the cells or the items, when the sum between first two

columns is greater than the value in the third column

5. **"%0+%1 > %2+%3"**, highlights the cells or items, when the sum between first two columns is greater than the sum between third and forth column.
6. **"%0+%1 >= 0 and (%2+%3)/2 < %4-5"**, highlights the cells or the items, when the sum between first two columns is greater than 0 and the half of the sum between third and forth columns is less than fifth column minus 5.
7. **"%0 startwith 'A'"** specifies the cells that starts with A
8. **"%0 endwith 'Bc'"** specifies the cells that ends with Bc
9. **"%0 contains 'aBc'"** specifies the cells that contains the aBc string
10. **"lower(%0) contains 'abc'"** specifies the cells that contains the abc, AbC, ABC, and so on
11. **"upper(%0)'"** retrieves the uppercase string
12. **"len(%0)>0'"** specifies the not blanks cells
13. **"len %0 = 0'"** specifies the blanks cells

The conditional format feature may change the cells and items as follows:

- Bold property. Bolds the cell or items
- Italic property. Indicates whether the cells or items should appear in italic.
- StrikeOut property. Indicates whether the cells or items should appear in strikeout.
- Underline property. Underlines the cells or items
- Font property. Changes the font for cells or items.
- BackColor property. Changes the background color for cells or items, supports skins as well.
- ForeColor property. Changes the foreground color for cells or items.

The following VB samples bolds all items when the sum between first two columns is greater than 0:

```
List1.ConditionalFormats.Add("%0+%1>0").Bold = True
```

The following C++ sample bolds all items when the sum between first two columns is greater than 0:

```
COleVariant vtEmpty;
m_List.GetConditionalFormats().Add( "%0+%1>0", vtEmpty ).SetBold( TRUE );
```

The following VB.NET sample bolds all items when the sum between first two columns is greater than 0:

```
AxList1.ConditionalFormats.Add("%0+%1>0").Bold = True
```

The following C# sample bolds all items when the sum between first two columns is greater

than 0:

```
axList1.ConditionalFormats.Add("%0+%1>0", null).Bold = true
```

The following VFP sample bolds all items when the sum between first two columns is greater than 0:

```
thisform.List1.ConditionalFormats.Add("%0+%1>0").Bold = .t.
```

property ConditionalFormat.Font as IFontDisp

Retrieves or sets the font for objects that match the criteria.

Type	Description
IFontDisp	A Font object that's applied to items or columns.

Use the Font property to change the font for items or columns that match the criteria. Use the Font property only, if you need to change to a different font.

You can change directly the font attributes, like follows:

- Bold property. Bolds the cell or items
- Italic property. Indicates whether the cells or items should appear in italic.
- StrikeOut property. Indicates whether the cells or items should appear in strikeout.
- Underline property. Underlines the cells or items

The following VB sample changes the font for ALL cells in the first column:

```
With List1.ConditionalFormats.Add("1")
   .ApplyTo = 0
   Set .Font = New StdFont
   With .Font
      .Name = "Comic Sans MS"
   End With
End With
```

property ConditionalFormat.ForeColor as Color

Retrieves or sets the foreground color for objects that match the condition.

Type	Description
Color	A color expression that indicates the foreground color for the object that match the criteria.

Use the ForeColor property to specify the foreground color for objects that match the criteria. Use the BackColor property to change the background color for items or cells in the column when a certain condition is met. Use the ClearForeColor method to remove the foreground color being set using previously the ForeColor property. If the ForeColor property is not set, it retrieves 0. The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column.

property ConditionalFormat.Italic as Boolean

Specifies whether the objects that match the condition should appear in italic.

Type	Description
Boolean	A boolean expression that indicates whether the objects should look in italic.

The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column. The following VB sample makes italic the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With List1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Italic = True
End With
```

The following C++ sample makes italic the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
COleVariant vtEmpty;
CConditionalFormat cf = m_list.GetConditionalFormats().Add( "%1+%2<%0", vtEmpty );
cf.SetItalic( TRUE );
cf.SetApplyTo( 1 );
```

The following VB.NET sample makes italic the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With AxList1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Italic = True
End With
```

The following C# sample makes italic the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
EXLISTLib.ConditionalFormat cf = axList1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Italic = true;
cf.ApplyTo = (EXLISTLib.FormatApplyToEnum)1;
```

The following VFP sample makes italic the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
with thisform.List1.ConditionalFormats.Add("%1+%2<%0")
   .Italic = .t.
   .ApplyTo = 1
endwith
```

property ConditionalFormat.Key as Variant

Checks whether the expression is syntactically correct.

Type	Description
Variant	A String expression that indicates the key of the element

The Key property indicates the key of the element. Use the Add method to specify a key at adding time. Use the Remove method to remove a formula giving its key.

property ConditionalFormat.StrikeOut as Boolean

Specifies whether the objects that match the condition should appear in strikeout.

Type	Description
Boolean	A Boolean expression that indicates whether the objects should appear in strikeout.

The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column. The following VB sample applies strikeout font attribute to cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With List1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C++ sample applies strikeout font attribute to cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
COleVariant vtEmpty;
CConditionalFormat cf = m_list.GetConditionalFormats().Add( "%1+%2<%0", vtEmpty );
cf.SetBold( TRUE );
cf.SetApplyTo( 1 );
```

The following VB.NET sample applies strikeout font attribute to cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With AxList1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C# sample applies strikeout font attribute to cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
EXLISTLib.ConditionalFormat cf = axList1.ConditionalFormats.Add("%1+%2<%0",null);
```

```
cf.Bold = true;
cf.ApplyTo = (EXLISTLib.FormatApplyToEnum)1;
```

The following VFP sample applies strikeout font attribute to cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
with thisform.List1.ConditionalFormats.Add("%1+%2<%0")
   .Bold = .t.
   .ApplyTo = 1
endwith
```

property ConditionalFormat.Underline as Boolean

Underlines the objects that match the condition.

Type	Description
Boolean	A boolean expression that indicates whether the objects are underlined.

The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column. The following VB sample underlines the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With List1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Underline = True
End With
```

The following C++ sample underlines the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
COleVariant vtEmpty;
CConditionalFormat cf = m_list.GetConditionalFormats().Add( "%1+%2<%0", vtEmpty );
cf.SetUnderline( TRUE );
cf.SetApplyTo( 1 );
```

The following VB.NET sample underlines the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With AxList1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Underline = True
End With
```

The following C# sample underlines the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
EXLISTLib.ConditionalFormat cf = axList1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Underline = true;
cf.ApplyTo = (EXLISTLib.FormatApplyToEnum)1;
```

The following VFP sample underlines the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
with thisform.List1.ConditionalFormats.Add("%1+%2<%0")
   .Underline = .t.
   .ApplyTo = 1
endwith
```

property ConditionalFormat.Valid as Boolean

Checks whether the expression is syntactically correct.

Type	Description
Boolean	A boolean expression that indicates whether the Expression property is valid.

Use the Valid property to check whether the Expression formula is valid. The conditional format is not applied to objects if expression is not valid, or the Enabled property is false. An empty expression is not valid. Use the Enabled property to disable applying the format to columns or items. Use the Remove method to remove an expression from ConditionalFormats collection.

ConditionalFormats object

The conditional formatting feature allows you to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. The ConditionalFormats collection holds a collection of ConditionalFormat objects. Use the ConditionalFormats property to access the control's ConditionalFormats collection .The ConditionalFormats collection supports the following properties and methods:

Name	Description
Add	Adds a new expression to the collection and returns a reference to the newly created object.
Clear	Removes all expressions in a collection.
Count	Returns the number of objects in a collection.
Item	Returns a specific expression.
Remove	Removes a specific member from the collection.

method ConditionalFormats.Add (Expression as String, [Key as Variant])

Adds a new expression to the collection and returns a reference to the newly created object.

Type	Description
Expression as String	A formal expression that indicates the formula being used when the format is applied. Please check the Expression property that shows the syntax of the expression that may be used. For instance, the "**%0 >= 10 and %1 > 67.23**" means all cells in the first column with the value less or equal than 10, and all cells in the second column with a value greater than 67.23
Key as Variant	A string or long expression that indicates the key of the expression being added. If the Key parameter is missing, by default, the current index in the ConditionalFormats collection is used.

Return	Description
ConditionalFormat	A ConditionalFormat object that indicates the newly format being added.

The conditional formatting feature allows you to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. Use the Add method to format cells or items based on values. Use the Add method to add new ConditionalFormat objects to the ConditionalFormats collection. By default, the ConditionalFormats collection is empty. A ConditionalFormat object indicates a formula and a format to apply to cells or items. The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to cells in the column. Use the Expression property to retrieve or set the formula. Use the Key property to retrieve the key of the object. Use the Refresh method to update the changes on the control's content.

The conditional format feature may change the cells and items as follows:

- Bold property. Bolds the cell or items
- Italic property. Indicates whether the cells or items should appear in italic.
- StrikeOut property. Indicates whether the cells or items should appear in strikeout.
- Underline property. Underlines the cells or items
- Font property. Changes the font for cells or items.
- BackColor property. Changes the background color for cells or items, supports skins as well.
- ForeColor property. Changes the foreground color for cells or items.

The following VB sample bolds all items when the sum between first two columns is greater than 0:

```
List1.ConditionalFormats.Add("%0+%1>0").Bold = True
```

The following VB sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With List1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C++ sample bolds all items when the sum between first two columns is greater than 0:

```
COleVariant vtEmpty;
m_list.GetConditionalFormats().Add( "%0+%1>0", vtEmpty ).SetBold( TRUE );
```

The following C++ sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
COleVariant vtEmpty;
CConditionalFormat cf = m_list.GetConditionalFormats().Add( "%1+%2<%0", vtEmpty );
cf.SetBold( TRUE );
cf.SetApplyTo( 1 );
```

The following VB.NET sample bolds all items when the sum between first two columns is greater than 0:

```
AxList1.ConditionalFormats.Add("%0+%1>0").Bold = True
```

The following VB.NET sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
With AxList1.ConditionalFormats.Add("%1+%2<%0")
   .ApplyTo = 1
   .Bold = True
End With
```

The following C# sample bolds all items when the sum between first two columns is greater

than 0:

```
axList1.ConditionalFormats.Add("%0+%1>0", null).Bold = true
```

The following C# sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
EXLISTLib.ConditionalFormat cf = axList1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXLISTLib.FormatApplyToEnum)1;
```

The following VFP sample bolds all items when the sum between first two columns is greater than 0:

```
thisform.List1.ConditionalFormats.Add("%0+%1>0").Bold = .t.
```

The following VFP sample bolds the cells in the second column (1), if the sum between second and third column (2) is less than the value in the first column (0):

```
with thisform.List1.ConditionalFormats.Add("%1+%2<%0")
   .Bold = .t.
   .ApplyTo = 1
endwith
```

method ConditionalFormats.Clear ()

Removes all expressions in a collection.

Type	Description

Use the Clear method to remove all objects in the collection. Use the Remove method to remove a particular object from the collection. Use the Enabled property to disable a conditional format.

property ConditionalFormats.Count as Long

Returns the number of objects in a collection.

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

Use the Item and Count property to enumerate the elements in the collection. Use the Expression property to get the expression of the format.

The following VB sample enumerates all elements in the ConditionalFormats collection:

```
Dim c As ConditionalFormat
For Each c In List1.ConditionalFormats
   Debug.Print c.Expression
Next
```

The following VB sample enumerates all elements in the ConditionalFormats collection:

```
Dim i As Integer
With List1.ConditionalFormats
   For i = 0 To .Count - 1
      Debug.Print .Item(i).Expression
   Next
End With
```

The following C++ sample enumerates all elements in the ConditionalFormats collection:

```
for ( long i = 0; i < m_list.GetConditionalFormats().GetCount(); i++ )
{
   CConditionalFormat cf = m_list.GetConditionalFormats().GetItem( COleVariant( i ) );
   OutputDebugString( cf.GetExpression() );
}
```

The following VB.NET sample enumerates all elements in the ConditionalFormats collection:

```
Dim c As EXLISTLib.ConditionalFormat
For Each c In AxList1.ConditionalFormats
   System.Diagnostics.Debug.Write(c.Expression)
Next
```

The following VB.NET sample enumerates all elements in the ConditionalFormats collection:

```
Dim i As Integer
With AxList1.ConditionalFormats
   For i = 0 To .Count - 1
      System.Diagnostics.Debug.Write(.Item(i).Expression)
   Next
End With
```

The following C# sample enumerates all elements in the ConditionalFormats collection:

```
foreach (EXLISTLib.ConditionalFormat c in axList1.ConditionalFormats)
   System.Diagnostics.Debug.Write(c.Expression);
```

The following C# sample enumerates all elements in the ConditionalFormats collection:

```
for (int i = 0; i < axList1.ConditionalFormats.Count; i++)
   System.Diagnostics.Debug.Write(axList1.ConditionalFormats[i].Expression);
```

The following VFP sample enumerates all elements in the ConditionalFormats collection:

```
with thisform.List1.ConditionalFormats
   for i = 0 to .Count - 1
      wait .Item(i).Expression
   next
endwith
```

property ConditionalFormats.Item (Key as Variant) as ConditionalFormat

Returns a specific expression.

Type	Description
Key as Variant	A long expression that indicates the index of the element being accessed, or a string expression that indicates the key of the element being accessed.
ConditionalFormat	A ConditionalFormat object being returned.

Use the Item and Count property to enumerate the elements in the collection. Use the Expression property to get the expression of the format. Use the Key property to get the key of the format.

The following VB sample enumerates all elements in the ConditionalFormats collection:

```
Dim c As ConditionalFormat
For Each c In List1.ConditionalFormats
   Debug.Print c.Expression
Next
```

The following VB sample enumerates all elements in the ConditionalFormats collection:

```
Dim i As Integer
With List1.ConditionalFormats
   For i = 0 To .Count - 1
      Debug.Print .Item(i).Expression
   Next
End With
```

The following C++ sample enumerates all elements in the ConditionalFormats collection:

```
for ( long i = 0; i < m_list.GetConditionalFormats().GetCount(); i++ )
{
   CConditionalFormat cf = m_list.GetConditionalFormats().GetItem( COleVariant( i ) );
   OutputDebugString( cf.GetExpression() );
}
```

The following VB.NET sample enumerates all elements in the ConditionalFormats collection:

```
Dim c As EXLISTLib.ConditionalFormat
```

```
For Each c In AxList1.ConditionalFormats
   System.Diagnostics.Debug.Write(c.Expression)
Next
```

The following VB.NET sample enumerates all elements in the ConditionalFormats collection:

```
Dim i As Integer
With AxList1.ConditionalFormats
   For i = 0 To .Count - 1
      System.Diagnostics.Debug.Write(.Item(i).Expression)
   Next
End With
```

The following C# sample enumerates all elements in the ConditionalFormats collection:

```
foreach (EXLISTLib.ConditionalFormat c in axList1.ConditionalFormats)
   System.Diagnostics.Debug.Write(c.Expression);
```

The following C# sample enumerates all elements in the ConditionalFormats collection:

```
for (int i = 0; i < axList1.ConditionalFormats.Count; i++)
   System.Diagnostics.Debug.Write(axList1.ConditionalFormats[i].Expression);
```

The following VFP sample enumerates all elements in the ConditionalFormats collection:

```
with thisform.List1.ConditionalFormats
   for i = 0 to .Count - 1
      wait .Item(i).Expression
   next
endwith
```

method ConditionalFormats.Remove (Key as Variant)

Removes a specific member from the collection.

Type	Description
Key as Variant	A Long or String expression that indicates the key of the element to be removed.

Use the Remove method to remove a particular object from the collection. Use the Enabled property to disable a conditional format. Use the Clear method to remove all objects in the collection.

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. The OleDragDrop event notifies your application that the user drags some data on the control.

property ExDataObject.Files as ExDataObjectFiles

Returns a DataObjectFiles collection, which in turn contains a list of all filenames used by a DataObject object.

Type	Description
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. The OleDragDrop event notifies your application that the user drags some data on the control.

method ExDataObject.GetData (Format as Integer)

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

Type	Description
Format as Integer	An 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 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. The OleDragDrop event notifies your application that the user drags some data on the control.

ExDataObjectFiles object

The ExDataObjectFiles contains a collection of filenames. The ExDataObjectFiles object is used in OLE Drag and drop events. In order to get the list of files used in drag and drop operations you have to use the Files property.

Name	Description
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. The OleStartDrag event notifies your application that the user starts dragging items.

method ExDataObjectFiles.Clear ()

Removes all file names in the collection.

Type	Description

Use the Clear method to remove all filenames from the collection.

property ExDataObjectFiles.Count as Long

Returns the number of file names in the collection.

Type	Description
Long	A long value that indicates the count of elements into collection.

You can use "for each" statements if you are going to enumerate the elements into ExDataObjectFiles collection.

property ExDataObjectFiles.Item (Index as Long) as String

Returns a specific file name given its index.

Type	Description
Index as Long	A long expression that indicates the filename's index.
String	A string value that indicates the filename.

method ExDataObjectFiles.Remove (Index as Long)

Removes a specific file name given its index into collection.

Type	Description
Index as Long	A long expression that indicates the index of filename into collection.

Use Clear method to remove all filenames.

Items object

The Items object contains a collection of items. Each item is identified by an index. Each item contains a collection of cells. The number of cells is determined by the number of Column objects in the control. To access the Items collection use Items property of the control. Using the Items collection you can add, remove or change the control items. The Items object supports the following properties and methods:

Name	Description
Add	Adds a new item, and returns the index of the newly created item.
Caption	Retrieves or sets the text displayed by a specific cell.
CaptionFormat	Specifies how the cell's caption is displayed.
CellBackColor	Retrieves or sets the cell's background color.
CellBold	Retrieves or sets a value that indicates whether the cell's caption should appear in bold.
CellChecked	Retrieves the cell's index that is checked on a specific radio group.
CellData	Retrieves or sets a value that indicates the extra data for a specific cell.
CellEnabled	Returns or sets a value that determines whether a cell can respond to user-generated events.
CellFont	Retrieves or sets the cell's font.
CellForeColor	Retrieves or sets the cell's foreground color.
CellHAlignment	Retrieves or sets a value that indicates the alignment of the cell's caption.
CellHasButton	Retrieves or sets a value indicating whether the cell has associated a push button.
CellHasCheckBox	Retrieves or sets a value indicating whether the cell has associated a checkbox.
CellHasRadioButton	Retrieves or sets a value indicating whether the cell has associated a radio button.
CellHyperLink	Specifies whether the cell's is highlighted when the cursor mouse is over the cell.
CellImage	Retrieves or sets a value that indicates the index of the cell's image into Images collection.
CellImages	Specifies an additional list of icons shown in the cell.

CellItalic	Retrieves or sets a value that indicates whether the cell's caption should appear in italic.
CellMerge	Retrieves or sets a value that indicates the index of the cell that's merged to.
CellPicture	Retrieves or sets a value that indicates the Picture object displayed by the cell.
CellPictureHeight	Retrieves or sets a value that indicates the height of the cell's picture.
CellPictureWidth	Retrieves or sets a value that indicates the width of the cell's picture.
CellRadioGroup	Retrieves or sets a value indicating the radio group where the cell is contained.
CellSingleLine	Retrieves or sets a value indicating whether the cell's caption is painted using one or more lines.
CellState	Retrieves or sets the cell's state. Has effect only for cells of check or radio types.
CellStrikeOut	Retrieves or sets a value that indicates whether the cell's caption should appear in strikeout.
CellToolTip	Retrieves or sets a value that indicates the cell's tooltip.
CellUnderline	Retrieves or sets a value that indicates whether the cell's caption is underlined.
CellVAlignment	Retrieves or sets a value that indicates how the cell's caption is vertically aligned.
ClearCellBackColor	Clears the cell's background color.
ClearCellForeColor	Clears the cell's foreground color.
ClearCellHAlignment	Clears the cell's alignment.
ClearItemBackColor	Clears the item's background color.
ClearItemForeColor	Clears the item's foreground color.
Count	Retrieves the number of items.
Edit	Edits a cell.
EnableItem	Returns or sets a value that determines whether a item can respond to user-generated events.
EnsureVisibleColumn	Ensures that a column fits the control client area.
EnsureVisibleItem	Ensures the given item is in the visible client area.
	Finds an item, looking for Caption in ColIndex colum. The

FindItem	searching starts at StartIndex item.
FindItemData	Finds the item giving its data.
FirstVisibleItem	Retrieves the index of the first visible item into control.
FocusItem	Retrieves the index of the item that has the focus.
FormatCell	Specifies the custom format to display the cell's content.
IsItemVisible	Checks if the specific item is in the visible client area.
ItemAllowSizing	Retrieves or sets a value that indicates whether a user can resize the item at run-time.
ItemBackColor	Retrieves or sets a background color for a specific item.
ItemBold	Retrieves or sets a value that indicates whether the item should appear in bold.
ItemBreak	Retrieves or sets a value that indicates whether the item is painted as a break line.
ItemData	Retrieves or sets a value that indicates the extra data for a specific item.
ItemFont	Retrieves or sets the item's font.
ItemForeColor	Retrieves or sets a foreground color for a specific item.
ItemHeight	Retrieves or sets the item's height.
ItemItalic	Retrieves or sets a value that indicates whether the item should appear in italic.
ItemMaxHeight	Retrieves or sets a value that indicates the maximum height when the item's height is variable.
ItemMinHeight	Retrieves or sets a value that indicates the minimum height when the item's height is sizing.
ItemPosition	Retrieves or sets a value that indicates the item's position.
ItemStrikeOut	Retrieves or sets a value that indicates whether the item should appear in strikeout.
ItemToVirtual	Gets the index of the virtual item giving the index of the item in the list.
ItemUnderline	Retrieves or sets a value that indicates whether the item is underlined.
LastVisibleItem	Retrieves the index of the last visible item.
MatchItemCount	Retrieves the number of items that match the filter.
NextVisibleItem	Retrieves the index of next visible item.

PrevVisibleItem	Retrieves the index of previous visible item.
Remove	Removes a specific item.
RemoveAll	Removes all items from the control.
RemoveSelection	Removes the selected items (including the descendents).
SelectableItem	Specifies whether the user can select the item.
SelectAll	Selects all items.
SelectCount	Retrieves the count of selected items.
SelectedItem	Retrieves the selected item's index given its index into selected items collection.
SelectItem	Selects or unselects a specific item.
Sort	Sorts a column.
SortableItem	Specifies whether the item is sortable.
UnselectAll	Unselects all items.
VirtualToItem	Gets the index of the item in the list giving the index of the virtual item.
VisibleCount	Retrieves the number of visible items.
VisibleItemCount	Retrieves the number of visible items.

method Items.Add ([Caption as Variant])

Adds a new item, and returns the index of the newly created item.

Type	Description
Caption as Variant	A string expression that indicates the caption for the cell in the first column, or a safe array that holds the captions for each column.

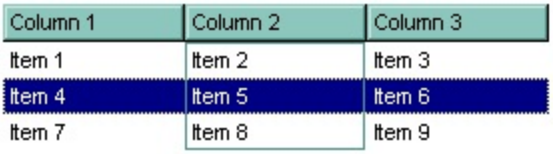
Return	Description
Long	A long value that indicate the index for the newly created item

The Add method adds a new item to the end of the list. If the control has more than a Column object, use the Caption property to set the caption for the rest of the cells. Use the CaptionFormat property to specify HTML tags in the caption, or a computed field. Use ItemPosition property to change the item's position. The AddItem event is fired once that a new item has been added. Use the PutItems method to load the control from an array. Use the DataSource property to bind the control to an ADO record set. The Add method is not available if the control is running in the virtual mode. Use the CellMerge property to combine two or more cells in a single cell. Use the BeginUpdate and EndUpdate methods to maintain performance while adding new columns and items. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample loads each item from a safe array:

```
With List1
  .BeginUpdate

  .Columns.Add "Column 1"
  .Columns.Add "Column 2"
  .Columns.Add "Column 3"

  With .Items
    .Add Array("Item 1", "Item 2", "Item 3")
    .Add Array("Item 4", "Item 5", "Item 6")
    .Add Array("Item 7", "Item 8", "Item 9")
  End With

  .EndUpdate
```

```
End With
```



The following VB sample add three columns and three items:

```
With List1
   .BeginUpdate

   .Columns.Add "Column 1"
   .Columns.Add "Column 2"
   .Columns.Add "Column 3"

   With .Items
      Dim i As Long
      i = .Add("Item 1")
      .Caption(i, 1) = "Item 2"
      .Caption(i, 2) = "Item 3"
      i = .Add("Item 4")
      .Caption(i, 1) = "Item 5"
      .Caption(i, 2) = "Item 6"
      i = .Add("Item 7")
      .Caption(i, 1) = "Item 8"
      .Caption(i, 2) = "Item 9"
   End With

   .EndUpdate
End With
```

The following C++ adds three columns and three items:

```
m_list.BeginUpdate();
CColumns columns = m_list.GetColumns();
CColumn column( V_DISPATCH( &columns.Add( "Column 1" ) ) );
columns.Add( "Column 2" );
columns.Add( "Column 3" );
```

```
CItems items = m_list.GetItems();
int i = items.Add( COleVariant("Item 1" ) );
items.SetCaption( i, COleVariant( long(1) ), COleVariant("SubItem 1.1" ) );
items.SetCaption( i, COleVariant( long(2) ), COleVariant("SubItem 1.2" ) );
i = items.Add( COleVariant("Item 2" ) );
items.SetCaption( i, COleVariant( long(1) ), COleVariant("SubItem 2.1" ) );
items.SetCaption( i, COleVariant( long(2) ), COleVariant("SubItem 2.2" ) );
i = items.Add( COleVariant("Item 3" ) );
items.SetCaption( i, COleVariant( long(1) ), COleVariant("SubItem 3.1" ) );
items.SetCaption( i, COleVariant( long(2) ), COleVariant("SubItem 3.2" ) );
m_list.EndUpdate();
```

The following VB.NET adds three columns and three items:

```
With AxList1
   .BeginUpdate()
   .Columns.Add("Column 1")
   .Columns.Add("Column 2")
   .Columns.Add("Column 3")
   With .Items
      Dim i As Integer = .Add("Item 1")
      .Caption(i, 1) = "SubItem 1.1"
      .Caption(i, 2) = "SubItem 1.2"
      i = .Add("Item 2")
      .Caption(i, 1) = "SubItem 2.1"
      .Caption(i, 2) = "SubItem 2.2"
      i = .Add("Item 3")
      .Caption(i, 1) = "SubItem 3.1"
      .Caption(i, 2) = "SubItem 3.2"
   End With
   .EndUpdate()
End With
```

The following C# adds three columns and three items:

```
axList1.BeginUpdate();
axList1.Columns.Add("Column 1");
axList1.Columns.Add("Column 2");
```

```
axList1.Columns.Add("Column 3");
EXLISTLib.Items items = axList1.Items;
int i = items.Add("Item 1");
items.set_Caption(i, 1,"SubItem 1.1");
items.set_Caption(i, 2,"SubItem 1.2");
i = items.Add("Item 2");
items.set_Caption(i, 1,"SubItem 2.1");
items.set_Caption(i, 2,"SubItem 2.2");
i = items.Add("Item 3");
items.set_Caption(i, 1,"SubItem 3.1");
items.set_Caption(i, 2,"SubItem 3.2");
axList1.EndUpdate();
```

The following VFP adds three columns and three items:

```
with thisform.List1
   .BeginUpdate()
   .Columns.Add("Column 1")
   .Columns.Add("Column 2")
   .Columns.Add("Column 3")
   With .Items
      Local i
      i = .Add("Item 1")
      .Caption(i, 1) = "SubItem 1.1"
      .Caption(i, 2) = "SubItem 1.2"
      i = .Add("Item 2")
      .Caption(i, 1) = "SubItem 2.1"
      .Caption(i, 2) = "SubItem 2.2"
      i = .Add("Item 3")
      .Caption(i, 1) = "SubItem 3.1"
      .Caption(i, 2) = "SubItem 3.2"
   EndWith
   .EndUpdate()
 endwith
```

property Items.Caption(Index as Long, ColIndex as Variant) as Variant

Retrieves or sets the text displayed by a specific cell.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Variant	A string expression that indicates the cell's caption.

The CellCaption property specifies the cell's content. To associate an user data for a cell you can use CellData property. Use the CaptionFormat property to use HTML tags in the cell's caption, or to specify a computed field . Use the ItemData property to associate an extra data to an item. To hide a column you have to use Visible property of the Column object. The Add method specifies also the caption for the first cell in the item. Use the **<img>** HTML tag to insert icons inside the cell's caption, if the CaptionFormat property is exHTML. For instance, the "some image <img>1</img> other image <img>2</img> rest of text", displays combined text and icons in the cell's caption. Use the Images method to load icons at runtime. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The Caption property is interpreted in different ways, based on the CaptionFormat property as follows:

- **exText**, the Caption property indicates just a plain text
- **exHTML**, the Caption property indicates a text that may include HTML format. The CaptionFormatEnum property indicates the built-in HTML tags that are supported.
- **exComputedField**, the Caption property indicates the formula to compute the cell. Use the ComputedField property to specify a formula for the entire column.

If the CaptionFormat property is exComputedField, the Caption property may may include variables, constants, operators or ( ) parenthesis. A variable is defined as %n, where n is the index of the column (zero based). For instance, the %0 indicates the first column, the %1, indicates the second column, and so on. A constant is a float expression (for instance, 23.45).

The property supports the following binary arithmetic operators:

- ***** (multiplicity operator), priority 5
- **/** (divide operator), priority 5
- **+** (addition operator), priority 4

- **-** (subtraction operator), priority 4

The property supports the following unary boolean operators:

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

The property supports the following binary boolean operators:

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

The property supports the following binary boolean operators, all these with the same priority 0 :

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

Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator. The property may be a combination of variables, constants and operators.

Samples:

1. "**1**", the cell displays 1
2. "**%0 + %1**", the cell displays the sum between cells in the first and second columns.
3. "**%0 + %1 - %2**", the cell displays the sum between cells in the first and second columns minus the third column.
4. "**(%0 + %1)*0.19**", the cell displays the sum between cells in the first and second columns multiplied with 0.19.
5. "**(%0 + %1 + %2)/3**", the cell displays the arithmetic average for the first three columns.
6. "**%0 + %1 < %2 + %3**", displays 1 if the sum between cells in the first two columns is less than the sum of third and forth columns.

property Items.CaptionFormat(Index as Long, ColIndex as Variant) as CaptionFormatEnum

Specifies how the cell's caption is displayed.

Type	Description
Index as Long	A long expression that specifies the index of item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
CaptionFormatEnum	A CaptionFormatEnum expression that indicates the cell's format.

The component supports built-in HTML format. That means that you can use HTML tags when displays the cell's caption . By default, the CaptionFormat property is exText. If the CaptionFormat is exText, the cell displays the Caption property without HTML format. If the CaptionFormat is exHTML, the cell displays the Caption property using the HTML tags specified in the CaptionFormatEnum type. Use the Def property to specify that all cells in the column display HTML format. Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property on exComputedField to indicate a computed field. Use the ComputedField property to specify a formula for the entire column.

The following VB sample adds a built-in HTML cell:

```
Dim i As Long
With List1.Items
   i = .Add("This is a built-in <b>HTML</b> cell.")
   .CaptionFormat(i, 0) = exHTML
End With
```

The following C++ sample adds a built-in HTML cell:

```
CItems items = m_list.GetItems();
int i = items.Add( COleVariant("This is a built-in <b>HTML</b> cell." ) );
items.SetCaptionFormat( i, COleVariant( long(0) ), 1 );
```

The following VB.NET sample adds a built-in HTML cell:

```
With AxList1.Items
```

```
    Dim i As Integer = .Add("This is a built-in <b>HTML</b> cell.")
    .CaptionFormat(i, 0) = EXLISTLib.CaptionFormatEnum.exHTML
End With
```

The following C# sample adds a built-in HTML cell:

```
EXLISTLib.Items items = axList1.Items;
int i = items.Add("This is a built-in <b>HTML</b> cell.");
items.set_CaptionFormat(i, 0, EXLISTLib.CaptionFormatEnum.exHTML );
```

The following VFP sample adds a built-in HTML cell:

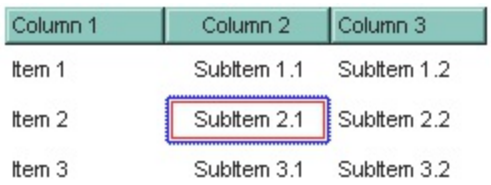
```
With thisform.List1.Items
    local i
    i = .Add("This is a built-in <b>HTML</b> cell.")
    .CaptionFormat(i, 0) = 1 && exHTML
EndWith
```

property Items.CellBackColor(Index as Long, ColIndex as Variant) as Color

Retrieves or sets the cell's background color.

Type	Description
Index as Long	A long expression that indicates the index of the item. If the Index is -1, the ClearItemBackColor clears the background color for all items.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Color	A color expression that indicates the cell's background color. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the CellBackColor property to set the cell's background color. Use the ItemBackColor property to change the item's background color. If the CellBackColor and CellForeColor were not used, the cell uses the control's background color, retrieved by the BackColor property. Use the CellForeColor property to change the cell's foreground color. Use the ClearCellBackColor property to clear the cell's background color when CellBackColor property was used. Use the Def(exCellBackColor) property to specify the background color for all cells in a column. You can use the CellBackColor property and the Add method to specify a different pattern on the cell's background. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.



The following VB sample changes the cell's visual appearance. The sample uses the " " skin to mark a cell:

```
With List1
  With .VisualAppearance
    .Add &H40, App.Path + "\cell.ebn"
```

```
    End With
    With .Items
        .CellBackColor(.FirstVisibleItem, 0) = &H40000000
    End With
End With
```

The following C++ sample changes the cell's appearance:

```
#include "Appearance.h"
#include "Items.h"
m_list.GetVisualAppearance().Add( 0x40,
COleVariant(_T("D:\\Temp\\ExList.Help\\cell.ebn")) );
m_list.GetItems().SetCellBackColor( COleVariant( m_list.GetItems().GetFirstVisibleItem() ),
COleVariant( long(0) ), 0x40000000 );
```

The following VB.NET sample changes the cell's appearance.

```
With AxList1
    With .VisualAppearance
        .Add(&H40, "D:\Temp\ExList.Help\cell.ebn")
    End With
    With .Items
        .CellBackColor(.FirstVisibleItem, 0) = &H40000000
    End With
End With
```

The following C# sample changes the cell's appearance.

```
axList1.VisualAppearance.Add(0x40, "D:\\Temp\\ExList.Help\\cell.ebn");
axList1.Items.set_CellBackColor(axList1.Items.FirstVisibleItem, 0, 0x40000000);
```

The following VFP sample changes the cell's appearance.

```
With thisform.List1
    With .VisualAppearance
        .Add(64, "D:\Temp\ExList.Help\cell.ebn")
    EndWith
    with .Items
        .DefaultItem = .FirstVisibleItem
```

```
        .CellBackColor(0,0) = 1073741824
    endwith
EndWith
```

property Items.CellBold(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value that indicates whether the cell's caption should appear in bold.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Boolean	A boolean expression that indicates whether the cell's caption is bolded.

Use the CellBold property to bold a cell. Use the ItemBold property to specify whether the item should appear in bold. Use the HeaderBold property of the Column object to bold the column's caption. Use the CellItalic, CellUnderline or CellStrikeOut property to apply different font attributes to the cell. Use the ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample bolds the cells in the first column

```
Dim h As Variant
With List1
   .BeginUpdate
   With .Items
      For Each h In List1.Items
         .CellBold(h, 0) = True
      Next
   End With
   .EndUpdate
End With
```

The following C++ sample bolds the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellBold( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
```

The following C# sample bolds the focused cell:

```
axList1.Items.set_CellBold(axList1.Items.FocusItem, 0, true);
```

The following VB.NET sample bolds the focused cell:

```
With AxList1.Items
   .CellBold(.FocusItem, 0) = True
End With
```

The following VFP sample bolds the focused cell:

```
with thisform.List1.Items
   .CellBold( .FocusItem, 0 ) = .t.
endwith
```

method Items.CellChecked (RadioGroup as Long, Index as Long, ColIndex as Long)

Retrieves the cell's index that is checked on a specific radio group.

Type	Description
RadioGroup as Long	A long expression that indicates the identifier of the radio group.
Index as Long	A long value that indicates the index of the item that contains the checked cell.
ColIndex as Long	A long value that indicates the index of the column that contains the checked cell.

A radio group contains a set of cells of radio types. Use the CellHasRadioButton property to set the cell of radio type. To change the state for a cell you can use the CellState property. To add or remove a cell to a given radio group you have to use CellHasRadioButton property. Use the CellRadioGroup property to add cells in the same radio group. The control fires the CellStateChanged event when the check box or radio button state is changed.

The following VB sample groups the radio cells in the first column, and displays the caption of the checked radio cell:

```
With List1
   .Columns(0).Def(exCellHasRadioButton) = True
   For Each i In .Items
      .Items.CellRadioGroup(i, 0) = 1234
   Next
End With
```

The following C++ sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
#include "Items.h"
COleVariant vtColumn( long(0) );
CItems items = m_list.GetItems();
m_list.BeginUpdate();
for ( long i = 0; i < items.GetCount(); i++ )
{
   items.SetCellHasRadioButton( i, vtColumn, TRUE );
```

```
    items.SetCellRadioGroup( i, vtColumn, 1234 );
}
m_list.EndUpdate();
```

The following VB.NET sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
With AxList1
    .BeginUpdate()
    .Columns(0).Def(EXLISTLib.DefColumnEnum.exCellHasRadioButton) = True
    With .Items
        Dim k As Integer
        For k = 0 To .Count - 1
            .CellRadioGroup(k, 0) = 1234
        Next
    End With
    .EndUpdate()
End With
```

The following C# sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
axList1.BeginUpdate();
EXLISTLib.Items items = axList1.Items;
axList1.Columns[0].set_Def(EXLISTLib.DefColumnEnum.exCellHasRadioButton, true);
for (int i = 0; i < items.Count; i++)
    items.set_CellRadioGroup(i, 0, 1234);
axList1.EndUpdate();
```

The following VFP sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
thisform.List1.BeginUpdate()
with thisform.List1.Items
    local i
    for i = 0 to .Count - 1
        .CellHasRadioButton( i,0 ) = .t.
        .CellRadioGroup(i,0) = 1234
    next
```

```
endwith
thisform.List1.EndUpdate()
```

property Items.CellData(Index as Long, ColIndex as Variant) as Variant

Retrieves or sets a value that indicates the extra data for a specific cell.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index or a string expression that indicates the column's name.
Variant	A VARIANT expression that indicates the cell's extra data.

Use the CellData to associate an extra data to your cell. Use ItemData when you need to associate an extra data with an item. The CellData value is not used by the control, it is only for user use. Use the Data property to assign an extra data to a column.

property Items.CellEnabled(Index as Long, ColIndex as Variant) as Boolean

Returns or sets a value that determines whether a cell can respond to user-generated events.

Type	Description
Index as Long	A long expression that indicates the item's index.
ColIndex as Variant	A long expression that indicates the column's index or a string expression that indicates the column's name.
Boolean	A boolean expression that indicates whether the cell is enabled or disabled.

Use the CellEnabled property to enable or disable a cell. If the cell is disabled, a check or radio button cannot be clicked in order to change its state. Use the Enabled property to disable the entire column. Use the EnableItem property to enable or disable an item. If an item is disabled or contained cells are disable no matter if the CellEnabled is True. Use the SelectableItem property to specify the user can select an item.

property Items.CellFont (Index as Long, ColIndex as Variant) as IFontDisp

Retrieves or sets the cell's font.

Type	Description
Index as Long	A long expression that specifies the index of item.
ColIndex as Variant	A long expression that specifies the index of column, a string expression that identifies the column's caption or column's key.
IFontDisp	A Font object being used by the cell.

By default, the CellFont property is nothing. If the CellFont property is noting, the cell uses the item's font. Use the CellFont and ItemFont properties to specify different fonts for cells or items. Use the CellBold, CellItalic, CellUnderline, CellStrikeout, ItemBold, ItemUnderline, ItemStrikeout, ItemItalic or CaptionFormat to specify different font attributes. Use the Refresh method to refresh the control's content on the fly. Use the BeginUpdate and EndUpdate methods if you are doing multiple changes, so no need for an update each time a change is done.

The following VB sample changes the font for the focused cell:

```
List1.BeginUpdate
With List1.Items
   .CellFont(.FocusItem, 0) = List1.Font
   With .CellFont(.FocusItem, 0)
      .Name = "Comic Sans MS"
      .Size = 10
      .Bold = True
   End With
End With
List1.EndUpdate
```

The following C++ sample changes the font for the focused cell:

```
#include "Items.h"
#include "Font.h"
m_list.BeginUpdate();
CItems items = m_list.GetItems();
long nItem = items.GetFocusItem();
```

```
COleVariant vtColumn( (long)0 );
items.SetCellFont( nItem, vtColumn, m_list.GetFont().m_lpDispatch );
COleFont font = items.GetCellFont( nItem, vtColumn );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_list.EndUpdate();
```

The following VB.NET sample changes the font for the focused cell:

```
AxList1.BeginUpdate()
With AxList1.Items
   .CellFont(.FocusItem, 0) = IFDH.GetIFontDisp(AxList1.Font)
   With .CellFont(.FocusItem, 0)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
AxList1.EndUpdate()
```

where the IFDH class is defined like follows:

```
Public Class IFDH
   Inherits System.Windows.Forms.AxHost

   Sub New()
      MyBase.New("")
   End Sub

   Public Shared Function GetIFontDisp(ByVal font As Font) As Object
      GetIFontDisp = AxHost.GetIFontFromFont(font)
   End Function

End Class
```

The following C# sample changes the font for the focused cell:

```
axList1.BeginUpdate();
axList1.Items.set_CellFont(axList1.Items.FocusItem, 0, IFDH.GetIFontDisp(axList1.Font));
stdole.IFontDisp spFont = axList1.Items.get_CellFont(axList1.Items.FocusItem, 0);
```

```
spFont.Name = "Comic Sans MS";
spFont.Bold = true;
axList1.EndUpdate();
```

where the IFDH class is defined like follows:

```
internal class IFDH : System.Windows.Forms.AxHost
{
    public IFDH() : base("")
    {
    }

    public static stdole.IFontDisp GetIFontDisp(System.Drawing.Font font)
    {
        return System.Windows.Forms.AxHost.GetIFontFromFont(font) as stdole.IFontDisp;
    }
}
```

The following VFP sample changes the font for the focused cell:

```
thisform.List1.Object.BeginUpdate()
with thisform.List1.Items
    .CellFont(.FocusItem,0) = thisform.List1.Font
    with .CellFont(.FocusItem,0)
        .Name = "Comic Sans MS"
        .Bold = .t.
    endwith
endwith
thisform.List1.Object.EndUpdate()
```

property Items.CellForeColor(Index as Long, ColIndex as Variant) as Color

Retrieves or sets the cell's foreground color.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index or a string expression that indicates the column's name.
Color	A color expression that indicates the cell's foreground color.

Use the CellForeColor property to change the cell's foreground color. Use the ItemForeColor property to change the item's foreground color. Use the CellBackColor property to change the cell's background color. If the CellForeColor and ItemForeColor were not used, the cell uses the control's ForeColor property. Use the ClearCellForeColor property to clear the cell's foreground color when the CellForeColor property is used. Use the Def(exCellForeColor) property to specify the foreground color for all cells in a column. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample changes the foreground color for the focused cell:

```
With List1.Items
   .CellForeColor(.FocusItem, 0) = vbRed
End With
```

In VB.NET or C# you require the following functions until the .NET framework will provide:

You can use the following VB.NET function:

```
   Shared Function ToUInt32(ByVal c As Color) As UInt32
      Dim i As Long
   i = c.R
   i = i + 256 * c.G
   i = i + 256 * 256 * c.B
   ToUInt32 = Convert.ToUInt32(i)
   End Function
```

You can use the following C# function:

```
private UInt32 ToUInt32(Color c)
{
    long i;
    i = c.R;
    i = i + 256 * c.G;
    i = i + 256 * 256 * c.B;
    return Convert.ToUInt32(i);
}
```

The following C# sample changes the foreground color for the focused cell:

```
axList1.Items.set_CellForeColor(axList1.Items.FocusItem, 0, ToUInt32(Color.Red) );
```

The following VB.NET sample changes the foreground color for the focused cell:

```
With AxList1.Items
    .CellForeColor(.FocusItem, 0) = ToUInt32(Color.Red)
End With
```

The following C++ sample changes the foreground color for the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellForeColor( items.GetFocusItem(), COleVariant( (long)0 ), RGB(255,0,0) );
```

The following VFP sample changes the foreground color for the focused cell:

```
with thisform.List1.Items
    .CellForeColor( .FocusItem, 0 ) = RGB(255,0,0)
endwith
```

property Items.CellHAlignment (Index as Long, ColIndex as Variant) as AlignmentEnum

Retrieves or sets a value that indicates the alignment of the cell's caption.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's key or the column's caption.
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the cell's caption.

The CellHAlignment property aligns a particular cell. Use the Alignment property of the Column object to align all the cells in the column. Use the CellVAlignment property to align vertically the caption of the cell, when the item displays its content using multiple lines. Use the ClearCellHAlignment method to clear the cell's alignment previously set by the CellHAlignment property. If the CellHAlignment property is not set, the Alignment property of the Column object indicates the cell's alignment.

The following VB sample right aligns the focused cell:

```
With List1.Items
   .CellHAlignment(.FocusItem, 0) = AlignmentEnum.RightAlignment
End With
```

The following C++ sample right aligns the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellHAlignment( items.GetFocusItem(), COleVariant( (long)0 ), 2
/*RightAlignment*/ );
```

The following VB.NET sample right aligns the focused cell:

```
With AxList1.Items
   .CellHAlignment(.FocusItem, 0) = EXLISTLib.AlignmentEnum.RightAlignment
End With
```

The following C# sample right aligns the focused cell:

```
axList1.Items.set_CellHAlignment(axList1.Items.FocusItem, 0,
EXLISTLib.AlignmentEnum.RightAlignment);
```

The following VFP sample right aligns the focused cell:

```
with thisform.List1.Items
   .CellHAlignment(.FocusItem,0) = 2 && RightAlignment
endwith
```

property Items.CellHasButton(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value indicating whether the cell has associated a push button.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Boolean	A boolean expression that indicates whether the cell has associated a push button.

The CellHasButton property specifies whether the cell display a button inside. When the cell's button is clicked the control fires CellButtonClick event. The caption of the push button is specified by the Caption property. Use the Def(exCellHasButton) property to assign buttons for all cells in the column. Use the Background property to change the visual appearance for cell's button.

The following VB sample assigns a button to the focused cell:

```
With List1.Items
   .CellHasButton(.FocusItem, 0) = True
End With
```

The following C++ sample assigns a button to the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellHasButton( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
```

The following VB.NET sample assigns a button to the focused cell:

```
With AxList1.Items
   .CellHasButton(.FocusItem, 0) = True
End With
```

The following C# sample assigns a button to the focused cell:

```
axList1.Items.set_CellHasButton(axList1.Items.FocusItem, 0, true);
```

The following VFP sample assigns a button to the focused cell:

```
with thisform.List1.Items
    .CellHasButton(.FocusItem,0) = .t.
endwith
```

property Items.CellHasCheckBox(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value indicating whether the cell has associated a checkbox.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Boolean	A boolean expression indicating whether the cell has associated a checkbox.

Use the CellHasCheckBox property to add a checkbox to the cell. Use the CellHasRadioButton property to add a radio button to the cell. A cell can displayed only a check box or a radio button at one time. If the user clicks a check or a radio button, the CellStateChanged event is fired. Use the CellState property to change the cell's state. Use the CheckImage property to change the check box appearance. Use the Def property to assign check boxes for all cells in the column. Use the CellImage property to add a single icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the FilterType property on exCheck to filter for checked or unchecked items.

The following VB sample adds a checkbox to the focused cell:

```
With List1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
End With
```

The following C++ sample adds a checkbox to the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellHasCheckBox( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
```

The following C# sample adds a checkbox to the focused cell:

```
axList1.Items.set_CellHasCheckBox(axList1.Items.FocusItem, 0, true);
```

The following VB.NET sample adds a checkbox to the focused cell:

```
With AxList1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
End With
```

The following VFP sample adds a checkbox to the focused cell:

```
with thisform.List1.Items
   .CellHasCheckBox(.FocusItem,0) = .t.
endwith
```

property Items.CellHasRadioButton(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value indicating whether the cell has associated a radio button.

Type	Description
Index as Long	A long expression that specifies the index of item.
ColIndex as Variant	A long expression that specifies the index of column, a string expression that identifies the column's caption or column's key.
Boolean	A boolean expression that specifies whether the cell contains radio button.

Retrieves or sets a value indicating whether the cell has associated a radio button or not. To change the state for a radio cell you have to use CellState property. The cell cannot display in the same time a radio and a check button. The control fires CellStateChanged event when the cell's state has been changed. To set the cell of check type you have call CellHasCheckBox property. To add or remove a cell to a given radio group you have to use CellRadioGroup property. Use the Def property to assign radio buttons for all cells in the column. Use the CellImage property to add a single icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the RadioImage property to change the radio button appearance.

The following VB sample enumerates all the cells in the first column and groups all of them in the same radio:

```
Dim h As Variant
List1.BeginUpdate
   With List1.Items
      For Each h In List1.Items
         .CellHasRadioButton(h, 0) = True
         .CellRadioGroup(h, 0) = 1234
      Next
   End With
List1.EndUpdate
```

```
or

Dim h As Variant
With List1
```

```
    .BeginUpdate
    .Columns(0).Def(exCellHasRadioButton) = True
    With List1.Items
        For Each h In List1.Items
            .CellRadioGroup(h, 0) = 1234
        Next
    End With
    .EndUpdate
End With
```

The following VB sample assigns a radio button to the focused cell:

```
With List1.Items
    .CellHasRadioButton(.FocusItem, 0) = True
    .CellRadioGroup(.FocusItem, 0) = 1234
End With
```

The following C++ sample assigns a radio button to the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellHasRadioButton( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
items.SetCellRadioGroup( items.GetFocusItem(), COleVariant( (long)0 ), 1234 );
```

The following VB.NET sample assigns a radio button to the focused cell:

```
With AxList1.Items
    .CellHasRadioButton(.FocusItem, 0) = True
    .CellRadioGroup(.FocusItem, 0) = 1234
End With
```

The following C# sample assigns a radio button to the focused cell:

```
axList1.Items.set_CellHasRadioButton(axList1.Items.FocusItem, 0, true);
axList1.Items.set_CellRadioGroup(axList1.Items.FocusItem, 0, 1234);
```

The following VFP sample assigns a radio button to the focused cell:

```
with thisform.List1.Items
    .CellHasRadioButton(.FocusItem,0) = .t.
```

```
    .CellRadioGroup(.FocusItem,0) = 1234
endwith
```

property Items.CellHyperLink (Index as Long, ColIndex as Variant) as Boolean

Specifies whether the cell's is highlighted when the cursor mouse is over the cell.

Type	Description
Index as Long	A long expression that specifies the index of item.
ColIndex as Variant	A long expression that specifies the index of column, a string expression that identifies the column's caption or column's key.
Boolean	A boolean expression that indicates whether the cell's is highlighted when the cursor mouse is over the cell.

A cell that has CellHyperLink property to True, is a cell of hyper link type. Use the CellHyperLink property to add hyper links to your control. Use the CellForeColor property to change the cell's foreground color. Use the HyperLinkColor property to change the color that's used by control when the cursor is over a cell of hyper link type. Use the <a> anchor element to mark hyperlinks in HTML captions.

property Items.CellImage (Index as Long, ColIndex as Variant) as Long

Retrieves or sets a value that indicates the index of the cell's image into Images collection.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Long	A long value that indicates the index of the cell's image into Images collection. The last 7 bits in the high significant byte of the long 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.

Use the CellImage to attach a single icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the CellPicture property to associate a larger picture to a cell. The cell's icon size is (16x16). If the index of image is not contained by the Images collection, no icon is displayed in the cell. If the cell contains an icon, the control fires CellImageClick event when the cell's icon is clicked. Use the ItemFromPoint property to retrieve the part of the control being clicked. Use the CellHasCheckBox property to add a check box to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the **<img>** HTML tag to insert icons inside the cell's caption, if the CaptionFormat property is exHTML. Use the FilterType property on exImage to filter items by icons. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample displays the first icon in the focused cell:

```
With List1.Items
   .CellImage(.FocusItem, 0) = 1
End With
```

The following C++ sample displays the first icon in the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellImage( items.GetFocusItem() , COleVariant( (long)0 ), 1 );
```

The following C# sample displays the first icon in the focused cell:

```
axList1.Items.set_CellImage(axList1.Items.FocusItem, 0, 1);
```

The following VB.NET sample displays the first icon in the focused cell:

```
With AxList1.Items
   .CellImage(.FocusItem, 0) = 1
End With
```

The following VFP sample displays the first icon in the focused cell:

```
with thisform.List1.Items
   .CellImage(.FocusItem,0) = 1
endwith
```

property Items.CellImages (Index as Long, ColIndex as Variant) as Variant

Specifies an additional list of icons shown in the cell.

Type	Description
Index as Long	A long expression that specifies the index of item.
ColIndex as Variant	A long expression that specifies the index of column, a string expression that identifies the column's caption or column's key.
Variant	A string expression that indicates the list of icons shown in the cell.

The CellImages property assigns multiple icons to a cell. The CellImage property assign a single icon to the cell. Instead if multiple icons need to be assigned to a single cell you have to use the CellImages property. The CellImages property takes a list of additional icons and display them in the cell. The list is separated by ',' and should contain numbers that represent indexes to Images list collection. Use the ItemFromPoint property to retrieve the part of the control being clicked. Use the CellHasCheckBox property to add a check box to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the **<img>** HTML tag to insert icons inside the cell's caption, if the CaptionFormat property is exHTML. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample assigns the first and third icon to the cell:

```
With List1.Items
    .CellImages(.ItemByIndex(0), 1) = "1,3"
End With
```

The following VB sample displays the index of icon being clicked:

```
Private Sub List1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim i As Long, h As HitTestInfoEnum, c As Long
  With List1
    i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
  End With
  If (i >= 0) Then
    If exHTCellIcon = (h And exHTCellIcon) Then
      Debug.Print "The index of icon being clicked is: " & (h And &HFFFF0000) / 65536
    End If
```

```
    End If
End Sub
```

The following C++ sample assigns the first and the third icon to the cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellImages( items.GetFocusItem(), COleVariant( (long)0 ), COleVariant( "1,3" ) );
```

The following C++ sample displays the index of icon being clicked:

```
#include "Items.h"
void OnMouseUpList1(short Button, short Shift, long X, long Y)
{
    CItems items = m_list.GetItems();
    long c = 0, hit = 0, h = m_list.GetItemFromPoint( X, Y, &c, &hit);
    if ( h >= 0 )
    {
        if ( ( hit & 0x44 /*exHTCellIcon*/ ) == 0x44 )
        {
            CString strFormat;
            strFormat.Format( "The index of icon being clicked is: %i\n", (hit >> 16) );
            OutputDebugString( strFormat );
        }
    }
}
```

The following VB.NET sample assigns the first and the third icon to the cell:

```
With AxList1.Items
    .CellImages(.FocusItem, 0) = "1,3"
End With
```

The following VB.NET sample displays the index of icon being clicked:

```
Private Sub AxList1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseUpEvent) Handles AxList1.MouseUpEvent
    With AxList1
        Dim i As Integer, c As Integer, hit As EXLISTLib.HitTestInfoEnum
```

```
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (i >= 0) Then
            Debug.WriteLine("The index of icon being clicked is: " & (hit And &HFFFF0000) /
65536)
        End If
    End With
End Sub
```

The following C# sample assigns the first and the third icon to the cell:

```
axList1.Items.set_CellImages(axList1.Items.FocusItem, 0, "1,3");
```

The following C# sample displays the index of icon being clicked:

```
private void axList1_MouseUpEvent(object sender,
AxEXLISTLib._IListEvents_MouseUpEvent e)
{
    int c = 0;
    EXLISTLib.HitTestInfoEnum hit;
    int i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if ((i >= 0))
    {
        if ((Convert.ToUInt32(hit) &
Convert.ToUInt32(EXLISTLib.HitTestInfoEnum.exHTCellIcon)) ==
Convert.ToUInt32(EXLISTLib.HitTestInfoEnum.exHTCellIcon))
        {
            string s = axList1.Items.get_Caption(i, c).ToString();
            s = "Cell: " + s + ", Icon's Index: " + (Convert.ToUInt32(hit) >> 16).ToString();
            System.Diagnostics.Debug.WriteLine(s);
        }
    }
}
```

The following VFP sample assigns the first and the third icon to the cell:

```
with thisform.List1.Items
    .CellImages(.FocusItem,0) = "1,3"
endwith
```

The following VFP sample displays the index of icon being clicked:

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

local c, hit, i
c = 0
hit = 0
with thisform.List1
   i = .ItemFromPoint( x, y, @c, @hit )
   if ( i >= 0 )
      if ( bitand( hit, 68 )= 68 )
         wait window nowait .Items.Caption( i, c ) + " " + Str( Int((hit - 68)/65536) )
      endif
   endif
endwith
```

Add the code to the MouseUp, MouseMove or MouseDown event

property Items.CellItalic(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value that indicates whether the cell's caption should appear in italic.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Boolean	A boolean expression that indicates whether the cell's caption should appear in italic.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample makes italic the focused cell:

```
With List1.Items
   .CellItalic(.FirstVisibleItem, 0) = True
End With
```

The following C++ sample makes italic the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellItalic( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
```

The following C# sample makes italic the focused cell:

```
axList1.Items.set_CellItalic(axList1.Items.FocusItem, 0, true);
```

The following VB.NET sample makes italic the focused cell:

```
With AxList1.Items
   .CellItalic(.FocusItem, 0) = True
End With
```

The following VFP sample makes italic the focused cell:

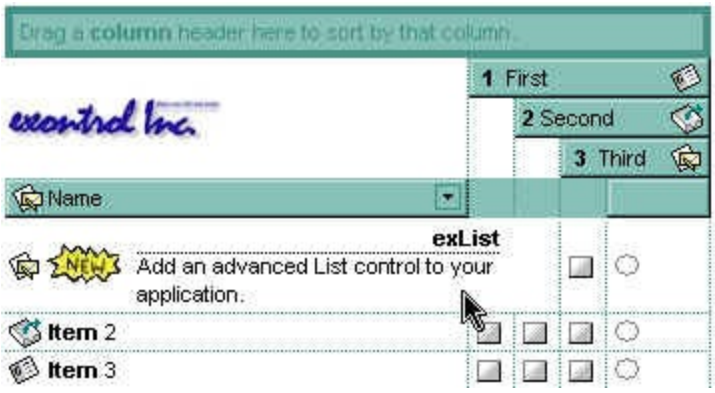
```
with thisform.List1.Items
    .CellItalic( .FocusItem, 0 ) = .t.
endwith
```

property Items.CellMerge(Index as Long, ColIndex as Variant) as Variant

Retrieves or sets a value that indicates the index of the cell that's merged to.

Type	Description
Index as Long	A long expression that indicates the index of the item where cells are merged.
ColIndex as Variant	A long expression that indicates the column's index
Variant	A long expression that indicates the index of the cell that's merged with, a safe array that holds the indexes of the cells being merged.

Use the CellMerge property to combine two or more cells in the same item in a single cell. The data of the source cell is displayed in the new larger cell. All the other cells' data is not lost. Use the CellMerge property to unmerge a single cell. Use the Add method to add new columns to the control. Use the Caption property to specify the caption of the cell. Use the SelectableItem property to specify whether the user can select an item. Use the BeginUpdate and EndUpdate methods to prevent control from painting while adding columns and items.



The following VB sample merges first three cells in a single one:

```
With List1.Items
   .CellMerge(.FirstVisibleItem, 0) = 1
   .CellMerge(.FirstVisibleItem, 0) = 2
End With
```

or

```
With List1.Items
   .CellMerge(.FirstVisibleItem, 0) = Array(1, 2)
End With
```

The following VB sample unmerges the first visible cell:

```
With List1.Items
   .CellMerge(.FirstVisibleItem, 0) = -1
End With
```

The following C++ sample merges first three cells in a single one:

```
CItems items = m_list.GetItems();
items.SetCellMerge( items.GetFirstVisibleItem(), COleVariant( long(0) ), COleVariant(
long(1) ) );
items.SetCellMerge( items.GetFirstVisibleItem(), COleVariant( long(0) ), COleVariant(
long(2) ) );
```

The following VB.NET sample merges first three cells in a single one:

```
With AxList1.Items
   .CellMerge(.FirstVisibleItem, 0) = 1
   .CellMerge(.FirstVisibleItem, 0) = 2
End With
```

or

```
With AxList1.Items
   Dim m() As Integer = {1, 2}
   .CellMerge(.FirstVisibleItem, 0) = m
End With
```

The following C# sample merges first three cells in a single one:

```
axList1.Items.set_CellMerge(axList1.Items.FirstVisibleItem, 0, 1);
axList1.Items.set_CellMerge(axList1.Items.FirstVisibleItem, 0, 2);
```

or

```
int[] m = {1,2};
axList1.Items.set_CellMerge(axList1.Items.FirstVisibleItem, 0, m);
```

The following VFP sample merges first three cells in a single one:

```
with thisform.List1.Items
```

```
   .CellMerge(.FirstVisibleItem,0) = 1
   .CellMerge(.FirstVisibleItem,0) = 2
endwith
```

property Items.CellPicture (Index as Long, ColIndex as Variant) as Variant

Retrieves or sets a value that indicates the Picture object displayed by the cell.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Variant	A Picture object that indicates the cell'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.

The control can associate to a cell a check or radio button, an icon, multiple icons, a picture and a caption. Use the CellPicture property to associate a picture to a cell. You can use the CellPicture property when you want to display images with different widths into a cell. Use the CellImage property to associate an icon from Images collection. Use the CellImages property to assign multiple icons to a cell. Use the CellHasCheckBox property to add a check box to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the ItemHeight property to enlarge the item's height. The CellPictureWidth and CellPictureHeight properties specifies the size of the area where the cell's picture is stretched.

The following VB sample loads a picture from a file:

```
With List1.Items
   .CellPicture(.FirstVisibleItem, 0) = "c:\winnt\logo.gif"
End With
```

The following sample loads the  picture whose base64 representation is:

```
"gBHJJGHA5MIqAAXAD3AENhozhpmhqZhrMhr/h0QGcQM0QTMQZkQf8QAESGcSM0STM
```

The following VB sample associates a picture to a cell by loading it from a base64 encoded string:

```
Dim s As String
s =
```

```
"gBHJJGHA5MIqAAXAD3AENhozhpmhqZhrMhr/h0QGcQM0QTMQZkQf8QAESGcSM0STM

s = s +
"JE6QQCj2UBhE0UAHGscgUEmIZXGqVQ1kcIg/CYcwIlEToBGiZwlHoPAYkEAYwBWHAUHGAB/

With List1.Items
   .CellPicture(.FocusItem, 0) = s
End With
```

The following C++ loads a picture from a file:

```
#include
BOOL LoadPicture( LPCTSTR szFileName, IPictureDisp** ppPictureDisp )
{
   BOOL bResult = FALSE;
   if ( szFileName )
   {
      OFSTRUCT of;
      HANDLE hFile = NULL;;
#ifdef _UNICODE
      USES_CONVERSION;
      if ( (hFile = (HANDLE)OpenFile( W2A(szFileName), &of;, OF_READ |
OF_SHARE_COMPAT)) != (HANDLE)HFILE_ERROR )
#else
      if ( (hFile = (HANDLE)OpenFile( szFileName, &of;, OF_READ | OF_SHARE_COMPAT)) !=
(HANDLE)HFILE_ERROR )
#endif
      {
         *ppPictureDisp = NULL;
         DWORD dwHighWord = NULL, dwSizeLow = GetFileSize( hFile, &dwHighWord; );
         DWORD dwFileSize = dwSizeLow;
         HRESULT hResult = NULL;
         if ( HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwFileSize) )
            if ( void* pvData = GlobalLock( hGlobal ) )
            {
               DWORD dwReadBytes = NULL;
               BOOL bRead = ReadFile( hFile, pvData, dwFileSize, &dwReadBytes;, NULL );
```

```
                GlobalUnlock( hGlobal );
                if ( bRead )
                {
                    CComPtr spStream;
                    _ASSERTE( dwFileSize == dwReadBytes );
                    if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal, TRUE, &spStream;) ) )
                        if ( SUCCEEDED( hResult = OleLoadPicture( spStream, 0, FALSE,
IID_IPictureDisp, (void**)ppPictureDisp ) ) )
                            bResult = TRUE;
                }
            }
        CloseHandle( hFile );
    }
  }
  return bResult;
}

IPictureDisp* pPicture = NULL;
if ( LoadPicture( "c:\\winnt\\zapotec.bmp", &pPicture; ) )
{
  COleVariant vtPicture;
  V_VT( &vtPicture; ) = VT_DISPATCH;
  pPicture->QueryInterface( IID_IDispatch, (LPVOID*)&V;_DISPATCH( &vtPicture; ) );
  CItems items = m_list.GetItems();
  items.SetCellPicture( items.GetFocusItem() , COleVariant(long(0)), vtPicture );
  pPicture->Release();
}
```

The following VB.NET sample loads a picture from a file:

```
With AxList1.Items
  .CellPicture(.FocusItem, 0) =
IPDH.GetIPictureDisp(Image.FromFile("c:\winnt\zapotec.bmp"))
End With
```

where the IPDH class is defined like follows:

```
Public Class IPDH
```

```
    Inherits System.Windows.Forms.AxHost

    Sub New()
        MyBase.New("")
    End Sub

    Public Shared Function GetIPictureDisp(ByVal image As Image) As Object
        GetIPictureDisp = AxHost.GetIPictureDispFromPicture(image)
    End Function

End Class
```

The following C# sample loads a picture from a file:

```
axList1.Items.set_CellPicture(axList1.Items.FocusItem, 0,
IPDH.GetIPictureDisp(Image.FromFile("c:\\winnt\\zapotec.bmp")));
```

where the IPDH class is defined like follows:

```
internal class IPDH : System.Windows.Forms.AxHost
{
    public IPDH() : base("")
    {
    }

    public static object GetIPictureDisp(System.Drawing.Image image)
    {
        return System.Windows.Forms.AxHost.GetIPictureDispFromPicture( image );
    }
}
```

The following VFP sample loads a picture from a file:

```
with thisform.List1.Items
    .CellPicture( .FocusItem, 0 ) = LoadPicture("c:\winnt\zapotec.bmp")
endwith
```

property Items.CellPictureHeight (Index as Long, ColIndex as Variant) as Long

Retrieves or sets a value that indicates the height of the cell's picture.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption or the column's key.
Long	A long expression that indicates the height of the cell's picture, or -1, if the property is ignored.

By default, the CellPictureHeight property is -1. Use the CellPicture property to assign a custom size picture to a cell. Use the CellImage or CellImages property to assign one or more icons to the cell. Use the CellPictureWidth property to specify the width of the cell's picture. The CellPictureWidth and CellPictureHeight properties specifies the size of the area where the cell's picture is stretched. If the CellPictureWidth and CellPictureHeight properties are -1 (by default), the cell displays the full size picture. If the CellPictureHeight property is greater than 0, it indicates the height of the area where the cell's picture is stretched. Use the ItemHeight property to specify the height of the item.

property Items.CellPictureWidth (Index as Long, ColIndex as Variant) as Long

Retrieves or sets a value that indicates the width of the cell's picture.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption or the column's key.
Long	A long expression that indicates the width of the cell's picture, or -1, if the property is ignored.

By default, the CellPictureWidth property is -1. Use the CellPicture property to assign a custom size picture to a cell. Use the CellImage or CellImages property to assign one or more icons to the cell. Use the CellPictureHeight property to specify the height of the cell's picture. The CellPictureWidth and CellPictureHeight properties specifies the size of the area where the cell's picture is stretched. If the CellPictureWidth and CellPictureHeight properties are -1 (by default), the cell displays the full size picture. If the CellPictureWidth property is greater than 0, it indicates the width of the area where the cell's picture is stretched.

property Items.CellRadioGroup(Index as Long, ColIndex as Variant) as Long

Retrieves or sets a value indicating the radio group where the cell is contained.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Long	A long value indicating the radio group where the cell is contained.

Use the CellRadioGroup property to add or remove a radio button from a group. In a radio group only one radio button can be checked. A radio cell cannot be contained by two different radio groups. Use the CellHasRadioButton property to add a radio button to a cell. When the cell's state is changed the control fires the CellStateChanged event. The CellState property specifies the cell's state. By default, when a cell of radio type is created the radio cell is not grouped to any of existent radio groups.

The following VB sample groups the radio cells in the first column, and displays the caption of the checked radio cell:

```
With List1
   .Columns(0).Def(exCellHasRadioButton) = True
   For Each i In .Items
      .Items.CellRadioGroup(i, 0) = 1234
   Next
End With
```

The following C++ sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
#include "Items.h"
COleVariant vtColumn( long(0) );
CItems items = m_list.GetItems();
m_list.BeginUpdate();
for ( long i = 0; i < items.GetCount(); i++ )
{
   items.SetCellHasRadioButton( i, vtColumn, TRUE );
```

```
    items.SetCellRadioGroup( i, vtColumn, 1234 );
}
m_list.EndUpdate();
```

The following VB.NET sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
With AxList1
    .BeginUpdate()
    .Columns(0).Def(EXLISTLib.DefColumnEnum.exCellHasRadioButton) = True
    With .Items
        Dim k As Integer
        For k = 0 To .Count - 1
            .CellRadioGroup(k, 0) = 1234
        Next
    End With
    .EndUpdate()
End With
```

The following C# sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
axList1.BeginUpdate();
EXLISTLib.Items items = axList1.Items;
axList1.Columns[0].set_Def(EXLISTLib.DefColumnEnum.exCellHasRadioButton, true);
for (int i = 0; i < items.Count; i++)
    items.set_CellRadioGroup(i, 0, 1234);
axList1.EndUpdate();
```

The following VFP sample groups the radio cells on the first column, and displays the caption of the checked radio cell:

```
thisform.List1.BeginUpdate()
with thisform.List1.Items
    local i
    for i = 0 to .Count - 1
        .CellHasRadioButton( i,0 ) = .t.
        .CellRadioGroup(i,0) = 1234
    next
```

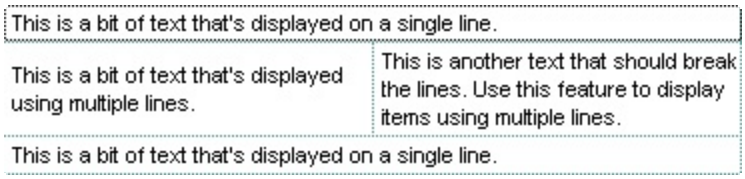
```
endwith
thisform.List1.EndUpdate()
```

property Items.CellSingleLine(Index as Long, ColIndex as Variant) as CellSingleLineEnum

Retrieves or sets a value indicating whether the cell's caption is painted using one or more lines.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
CellSingleLineEnum	A CellSingleLineEnum expression that indicates whether the cell displays its caption using one or more lines.

By default, the CellSingleLine property is exCaptionSingleLine / True, which indicates that the cell's caption is displayed on a single line. Use the Def(exCellSingleLine) property to specify that all cells in the column display their content using multiple lines. The control can displays the cell's caption using more lines, if the CellSingleLine property is exCaptionWordWrap or exCaptionBreakWrap. The CellSingleLine property wraps the cell's caption so it fits in the cell's client area. If the text doesn't fit the cell's client area, the height of the item is increased or decreased. When the CellSingleLine is exCaptionWordWrap / exCaptionBreakWrap / False, the height of the item is computed based on each cell caption. *If the CellSingleLine property is* exCaptionWordWrap / exCaptionBreakWrap / *False, changing the ItemHeight property has no effect.* Use the ItemMaxHeight property to specify the maximum height of the item when its height is variable. Use the CellVAlignment property to align vertically a cell.



If using the CellSingleLine / Def(exCellSingleLine) property, we recommend to set the ScrollBySingleLine property on True so all items can be scrolled.

The following VB sample displays the caption of the focused cell using multiple lines:

```
With List1.Items
   .CellSingleLine(.FocusItem, 0) = True
End With
```

The following C++ sample displays the caption of the focused cell using multiple lines:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellSingleLine( items.GetFocusItem() , COleVariant( long(0) ), FALSE );
```

The following VB.NET sample displays the caption of the focused cell using multiple lines:

```
With AxList1.Items
   .CellSingleLine(.FocusItem, 0) = False
End With
```

The following C# sample displays the caption of the focused cell using multiple lines:

```
axList1.Items.set_CellSingleLine(axList1.Items.FocusItem, 0, false);
```

The following VFP sample displays the caption of the focused cell using multiple lines:

```
with thisform.List1.Items
   .CellSingleLine( .FocusItem, 0 ) = .f.
endwith
```

property Items.CellState(Index as Long, ColIndex as Variant) as Long

Retrieves or sets the cell's state.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Long	A long value that indicates the cell's state. 0 - Unchecked, 1 - Checked, (2 - Partial Checked, only for cells of check type).

Use the CellState property to change the cell's state. The CellState property has effect only for check and radio cells. Use the CellHasCheckBox property to assign a check box to a cell. Use the CellHasRadioButton property to add a radio button to a cell. The control fires the CellStateChanged event when user changes the cell's state. Use the CheckImage property to change the check box appearance. Use the RadioImage property to change the radio button appearance. Use the FilterType property on exCheck to filter for checked or unchecked items.

The following VB sample adds a check box that's checked to the focused cell:

```
With List1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
   .CellState(.FocusItem, 0) = 1
End With
```

The following C++ sample adds a check box that's checked to the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
COleVariant vtColumn( long(0) );
long nItem = items.GetFocusItem();
items.SetCellHasCheckBox( nItem, vtColumn, TRUE );
items.SetCellState( nItem, vtColumn, 1 );
```

The following VB.NET sample adds a check box that's checked to the focused cell:

```
With AxList1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
```

```
    .CellState(.FocusItem, 0) = 1
End With
```

The following C# sample adds a check box that's checked to the focused cell:

```
axList1.Items.set_CellHasCheckBox(axList1.Items.FocusItem, 0, true);
axList1.Items.set_CellState(axList1.Items.FocusItem, 0, 1);
```

The following VFP sample adds a check box that's checked to the focused cell:

```
with thisform.List1.Items
    .CellHasCheckBox( .FocusItem, 0 ) = .t.
    .CellState( .FocusItem,0 ) = 1
endwith
```

property Items.CellStrikeOut(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value that indicates whether the cell's caption should appear in strikeout.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Boolean	A boolean expression that indicates whether the cell is displayed with a horizontal line through it.

If the CellStrikeOut property is True, the cell's font is displayed with a horizontal line through it. Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption.

The following VB sample draws a horizontal line through the caption of the cell that has the focus:

```
With List1.Items
   .CellStrikeOut(.FirstVisibleItem, 0) = True
End With
```

The following C++ sample draws a horizontal line through the caption of the cell that has the focus:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellStrikeOut( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
```

The following C# sample draws a horizontal line through the caption of the cell that has the focus:

```
axList1.Items.set_CellStrikeOut(axList1.Items.FocusItem, 0, true);
```

The following VB.NET sample draws a horizontal line through the caption of the cell that has the focus:

```
With AxList1.Items
    .CellStrikeOut(.FocusItem, 0) = True
End With
```

The following VFP sample draws a horizontal line through the caption of the cell that has the focus:

```
with thisform.List1.Items
    .CellStrikeOut( .FocusItem, 0 ) = .t.
endwith
```

property Items.CellToolTip(Index as Long, ColIndex as Variant) as String

Retrieves or sets a value that indicates the cell's tooltip.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
String	A string expression that indicates the cell's tooltip.

Use the CellToolTip property to associate a tooltip to a cell. By default, the CellToolTip property is "..." (three dots). If the CellToolTip property is "..." the control shows a tooltip that displays the cell's caption if the cell's caption doesn't fit the cell's client area. Use the Caption property to change the cell's caption. Use the ToolTipWidth property to specify the width of the tooltip window. 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 ShowToolTip method to display a custom tooltip.

The tooltip supports the foolowing 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.

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

  - exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
  - e64, encodes in BASE64 the HTML text to be shown once the user clicks the

anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu " that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY string encodes the "<fgcolor 808080>show lines<a>-</fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part

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

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

gradient-center

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

outlined

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

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

shadow

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

outline anti-aliasing

property Items.CellUnderline(Index as Long, ColIndex as Variant) as Boolean

Retrieves or sets a value that indicates whether the cell's caption is underlined.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Boolean	A boolean expression that indicates whether the cell's caption is underlined.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample underlines the focused cell:

```
With List1.Items
   .CellUnderline(.FirstVisibleItem, 0) = True
End With
```

The following C++ sample underlines the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellUnderline( items.GetFocusItem(), COleVariant( (long)0 ), TRUE );
```

The following C# sample underlines the focused cell:

```
axList1.Items.set_CellUnderline(axList1.Items.FocusItem, 0, true);
```

The following VB.NET sample underlines the focused cell:

```
With AxList1.Items
   .CellUnderline(.FocusItem, 0) = True
End With
```

The following VFP sample underlines the focused cell:

```
with thisform.List1.Items
    .CellUnderline( .FocusItem, 0 ) = .t.
endwith
```

property Items.CellVAlignment (Index as Long, ColIndex as Variant) as VAlignmentEnum

Retrieves or sets a value that indicates how the cell's caption is vertically aligned.

Type	Description
Index as Long	A long expression that specifies the index of the item.
ColIndex as Variant	A long expression that specifies the index of column, a string expression that identifies the column's caption or column's key.
VAlignmentEnum	A VAlignmentEnum expression that indicates the vertical cell's alignment

The CellVAlignment property aligns vertically the cell's caption. Use the ItemHeight property to specify the item's height. The Alignment property aligns horizontally the cells in a column. The CellHAlignment property aligns a particular cell. the CellSingleLine property to specify whether a cell uses single or multiple lines. Use the Def(exCellVAlignment) property to specify the same vertical alignment for the entire column.

The following VB sample aligns the focused cell to the bottom:

```
With List1.Items
   .CellVAlignment(.FocusItem, 0) = VAlignmentEnum.BottomAlignment
End With
```

The following C++ sample right aligns the focused cell:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetCellVAlignment( items.GetFocusItem() , COleVariant( (long)0 ), 2
/*BottomAlignment*/ );
```

The following VB.NET sample right aligns the focused cell:

```
With AxList1.Items
   .CellVAlignment(.FocusItem, 0) = EXLISTLib.VAlignmentEnum.BottomAlignment
End With
```

The following C# sample right aligns the focused cell:

```
axList1.Items.set_CellVAlignment(axList1.Items.FocusItem, 0,
```

```
EXLISTLib.VAlignmentEnum.BottomAlignment);
```

The following VFP sample right aligns the focused cell:

```
with thisform.List1.Items
   .CellVAlignment(.FocusItem,0) = 2 && BottomAlignment
endwith
```

method Items.ClearCellBackColor (Index as Long, ColIndex as Variant)

Clears the cell's background color.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption, or column's key.

The ClearCellBackColor method clears the cell's background color when the CellBackColor property was used. Use the BackColor property to specify the control's background color. Use the ItemBackColor property to specify the item's background color.

method Items.ClearCellForeColor (Index as Long, ColIndex as Variant)

Clears the cell's foreground color.

Type	Description
Index as Long	A long expression that indicates the index of the item
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption or column's key.

The ClearCellForeColor method clears the cell's foreground color when CellForeColor property was used. Use the ForeColor property to specify the control's foreground color. Use the ItemForeColor property to specify the item's foreground color.

method Items.ClearCellHAlignment (Index as Long, ColIndex as Variant)

Clears the cell's alignment.

Type	Description
Index as Long	A long expression that indicates the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's key or the column's caption.

Use the ClearCellHAlignment method to clear the alignment of the cell's caption previously set using the CellHAlignment property. If the CellHAlignment property is not called, the Alignment property of the Column object specifies the alignment of the cell's caption.

method Items.ClearItemBackColor (Index as Long)

Clears the item's background color.

Type	Description
Index as Long	A long expression that identifies the item's index.

The ClearItemBackColor method clears the item's background color when the ItemBackColor property is used. Use the BackColor property to specify the control's background color. Use the CellBackColor property to change the cell's background color.

method Items.ClearItemForeColor (Index as Long)

Clears the item's foreground color.

Type	Description
Index as Long	A long expression that indicates the index of the item.

The ClearItemForeColor method clears the item's foreground color when the ItemForeColor property is used. Use the ForeColor property to specify the control's foreground color. Use the CellForeColor property to change the cell's foreground color.

property Items.Count as Long

Retrieves the number of items.

Type	Description
Long	A long expression that indicates the items count.

Use the Count property to count the items into the Items collection. Use the VisibleCount property to get the count of visible items. Use the FirstVisibleItem property to get the index of the first visible item. Use the NextVisibleItem property to get the index of the next visible item. Use the SortOrder property to sort a column. Use the ItemPosition property to specify the position of the item.

The following VB sample displays all items in the list:

```
With List1.Items
   For i = 0 To .Count - 1
      Debug.Print .Caption(i, 0)
   Next
End With
```

The following C++ sample displays all items in the list:

```
CItems items = m_list.GetItems();
for ( long i = 0; i < items.GetCount(); i++ )
{
   CString strCaption = V2S( &items.GetCaption( i, COleVariant( long(0) ) ) );
   OutputDebugString( strCaption );
}
```

where the V2S function converts a VARIANT expression to a string expression:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
```

```
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}
```

The following VB.NET sample displays all items in the list:

```
With AxList1.Items
    Dim i As Integer
    For i = 0 To .Count - 1
        Debug.WriteLine(.Caption(i, 0))
    Next
End With
```

The following C# sample displays all items in the list:

```
for (int i = 0; i < axList1.Items.Count; i++)
{
    object cell = axList1.Items.get_Caption(i, 0);
    System.Diagnostics.Debug.WriteLine(cell != null ? cell.ToString() : "");
}
```

The following VFP sample displays all items in the list:

```
With thisform.List1.Items
    For i = 0 To .Count - 1
        wait window nowait .Caption(i, 0)
    Next
EndWith
```

method Items.Edit (Index as Long, ColIndex as Variant)

Edits a cell.

Type	Description
Index as Long	A long expression that identifies the index of item being edited.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that specifies the column's caption or the column's key.

Use the Edit method to programmatically edit a cell. The BeforeCellEdit event is fired when a cell starts to be edited. The AfterCellEdit event is fired when edit operation ends. The CancelCellEdit event occurs if the user cancels the edit operation. Use the SelStart, SelLength properties to specify the coordinates of the text being selected when edit starts. The edit operation starts only if the control's AllowEdit property is True.

The following VB sample starts editing a cell as soon as user clicks a cell:

```
Private Sub List1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim c As Long, i As Long, hit As HitTestInfoEnum
   With List1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
      If (i >= 0) Then
         .AllowEdit = True
         .Items.Edit i, c
      End If
   End With
End Sub
```

The following C++ sample starts editing a cell as soon as user clicks a cell:

```
void OnMouseDownList1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
   if ( i >= 0 )
   {
      CItems items = m_list.GetItems();
      m_list.SetAllowEdit( TRUE );
      items.Edit( i, COleVariant( c ) );
```

```
    }
}
```

The following VB.NET sample starts editing a cell as soon as user clicks a cell:

```
Private Sub AxList1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseDownEvent) Handles AxList1.MouseDownEvent
    Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
    Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
    If (i >= 0) Then
        AxList1.AllowEdit = True
        With AxList1.Items
            .Edit(i, c)
        End With
    End If
End Sub
```

The following C# sample starts editing a cell as soon as user clicks a cell:

```
private void axList1_MouseDownEvent(object sender,
AxEXLISTLib._IListEvents_MouseDownEvent e)
{
    EXLISTLib.HitTestInfoEnum hit;
    int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i >= 0)
    {
        axList1.AllowEdit = true;
        axList1.Items.Edit(i, c);
    }
}
```

The following VFP sample starts editing a cell as soon as user clicks a cell:

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

local c, i, hit
With thisform.List1
    c = 0
```

```
    hit = 0
    i = .ItemFromPoint(x, y, @c, @hit)
    If (i >= 0)
        .AllowEdit = .t.
        .Items.Edit( i,c)
    EndIf
EndWith
```

Put the code in the MouseDown event.

property Items.EnableItem(Index as Long) as Boolean

Returns or sets a value that determines whether a item can respond to user-generated events.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	A boolean expression that indicates whether the item is enabled or disabled.

Use the EnableItem property to disable an item. A disabled item looks grayed and it is selectable. Use the SelectableItem property to specify the user can select an item. Once that an item is disabled all the cells of the item are disabled, so CellEnabled property has no effect. To disable a column you can use Enabled property of a Column object.

method Items.EnsureVisibleColumn (ColIndex as Variant)

Ensures that a column fits the control client area.

Type	Description
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.

The EnsureVisibleColumn method ensures that the given column fits the control's client area. The EnsureVisibleColumn method has no effect if the column is hidded. Use the Visible property to show or hide a column. Use the Position property to change the column's position. Use the EnsureVisibleItem method to ensure that an item fits the control's client area. Use the ScrollBars property to hide the control's scroll bars.

method Items.EnsureVisibleItem (Index as Long)

Ensures the given item is in the visible client area.

Type	Description
Index as Long	A long expression that indicates the index of the item.

The control scrolls the list until the item fits the client area. Use the EnsureVisibleColumn property to ensure that a cell fits the client area. Use the SelectItem property to select an item. Use the SelectableItem property to specify whether the user can select an item. Use the ScrollPos property to scroll the control's content.

property Items.FindItem (Caption as Variant, [ColIndex as Variant], [StartIndex as Variant]) as Long

Finds an item, looking for Caption in ColIndex column. The searching starts at StartIndex item.

Type	Description
Caption as Variant	A Variant expression that indicates the caption that is searched for.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
StartIndex as Variant	A long value that indicates the index of item from where the searching starts.
Long	A long value that indicates the index of item found.

Use the FindItem to search for an item. Finds a control's item that matches Caption(Item, ColIndex) = Caption. The StartIndex parameter indicates the index from where the searching starts. If it is missing, the searching starts from the item with the 0 index. Use the AutoSearch property to enable incremental search feature within the column.

The following VB sample looks and selects the "Item 2":

```
With List1.Items
   Dim i As Long
   i = .FindItem("Item 2",0)
   If (i >= 0) Then
      .SelectItem(i) = True
   End If
End With
```

The following C++ sample looks and selects the "Item 2":

```
CItems items = m_list.GetItems();
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
int i = items.GetFindItem( COleVariant("Item 2"), COleVariant( long(0) ), vtMissing );
if ( i >= 0 )
   items.SetSelectItem( i, TRUE );
```

The following VB.NET sample looks and selects the "Item 2":

```
With AxList1.Items
   Dim i As Integer = .FindItem("Item 2", 0)
   If (i >= 0) Then
      .SelectItem(i) = True
   End If
End With
```

The following C# sample looks and selects the "Item 2":

```
int i = axList1.Items.get_FindItem("Item 2", 0, null);
if (i >= 0)
   axList1.Items.set_SelectItem(i, true);
```

The following VFP sample looks and selects the "Item 2":

```
With thisform.List1.Items
   local i
   i = .FindItem("Item 2",0)
   If (i >= 0) Then
      .SelectItem(i) = .t.
   EndIf
EndWith
```

property Items.FindItemData (UserData as Variant, [StartIndex as Variant]) as Long

Finds the item giving its data.

Type	Description
UserData as Variant	A variant value that indicates the value being searched
StartIndex as Variant	A long expression that indicates the index of the item where the searching starts
Long	A long expression that indicates the index of the item found, or -1 if the item is not found.

Use the FindItemData property to search for an item giving its extra-data. Use the ItemData property to associate an extra data to an item. Use the FindItem property to locate an item given its caption.

property Items.FirstVisibleItem as Long

Retrieves the index of the first visible item into control.

Type	Description
Long	A long expression that indicates the index of the first visible item into control.

Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items as they are listed. The LastVisibleItem property retrieves the index of last item that fits the client area. Use the PrevVisibleItem property to get the previous item. Use the SortOrder property to sort a column. Use the ItemPosition property to change the item's position. Use the ItemFromPoint property to get the item from cursor. Use the GetItems property to get the list of items.

The following VB sample displays the items as they are listed:

```
With List1.Items
   Dim i As Long
   i = .FirstVisibleItem
   While (i >= 0)
      Debug.Print .Caption(i, 0)
      i = .NextVisibleItem(i)
   Wend
End With
```

The following C++ sample displays the items as they are listed:

```
CItems items = m_list.GetItems();
long i = items.GetFirstVisibleItem();
while ( i >= 0 )
{
   CString strCaption = V2S( &items.GetCaption( i, COleVariant( long(0) ) ) );
   OutputDebugString( strCaption );
   i = items.GetNextVisibleItem( i );
}
```

The following VB.NET sample displays the items as they are listed:

```
With AxList1.Items
```

```
    Dim i As Integer = .FirstVisibleItem
    While (i >= 0)
        Debug.WriteLine(.Caption(i, 0))
        i = .NextVisibleItem(i)
    End While
End With
```

The following C# sample displays the items as they are listed:

```
int i = axList1.Items.FirstVisibleItem;
while (i >= 0)
{
    object cell = axList1.Items.get_Caption(i, 0);
    System.Diagnostics.Debug.WriteLine(cell != null ? cell.ToString() : "");
    i = axList1.Items.get_NextVisibleItem(i);
}
```

The following VFP sample displays the items as they are listed:

```
With thisform.List1.Items
    local i
    i = .FirstVisibleItem
    do While (i >= 0)
        wait window .Caption(i, 0)
        i = .NextVisibleItem(i)
    enddo
EndWith
```

property Items.FocusItem as Long

Retrieves the index of the item that has the focus.

Type	Description
Long	A long value that indicates the index of the item that has the focus.

The FocusItem property specifies the index of the focused item. If there is no focused item the FocusItem property retrieves -1. At one moment, only one item can be focused. When the selection is changed the focused item is changed too. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. Use the ShowFocusRect property to indicate whether the control draws a marking rectangle around the focused item. You can change the focused item, by selecting a new item using the SelectItem method. If the items is not selectable, it is not focusable as well. Use the SelectableItem property to specify whether an item is selectable/focusable.

property Items.FormatCell(Index as Long, [ColIndex as Variant]) as String

Specifies the custom format to display the cell's content.

Type	Description
Index as Long	A Long expression that specifies the index of the item.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's key or the column's caption.
String	A string expression that indicates the format to be applied on the cell's value, including HTML formatting, if the cell supports it.

By default, the FormatCell property is empty. The format is being applied if valid (not empty, and syntactically correct). The expression may be a combination of variables, constants, strings, dates and operators, and value. The *value* operator gives the value to be formatted. A string is delimited by ", ` or ' characters, and inside they can have the starting character preceded by \ character, ie "\"This is a quote\"". A date is delimited by # character, ie #1/31/2001 10:00# means the January 31th, 2001, 10:00 AM. The FormatColumn property applies the predefined format for all cells in the columns. The Caption property indicates the cell's caption.

The CellValue property of the cell is being shown as:

- formatted using the FormatCell property, if it is valid
- formatted using the FormatColumn property, if it is valid

In other words, all cells applies the format of the FormatColumn property, excepts the cells with the FormatCell property being set. If the cell belongs to a column with the FireFormatColumn property on True, the Value parameter of the FormatColumn event shows the newly caption for the cell to be shown.

For instance:

- the "*currency(value)*" displays the column using the current format for the currency ie, 1000 gets displayed as $1,000.00
- the "*longdate(date(value))*" converts the value to a date and gets the long format to display the date in the column, ie #1/1/2001# displays instead Monday, January 01, 2001
- the "*'' + ((0:=proper(value)) left 1) + '' + (=:0 mid 2)*" converts the name to proper, so the first letter is capitalized, bolds the first character, and let unchanged the rest, ie a "mihai filimon" gets displayed "**M**ihai Filimon".
- the "*len(value) ? ((0:=dbl(value)) < 10 ? '<fgcolor=808080><font ;7>' : '') +*

currency(=:0)" displays the cells that contains not empty daya, the value in currency format, with a different font and color for values less than 10, and bolded for those that are greater than 10, as can see in the following screen shot in the column (A+B+C):

Name	A	B	C	A+B+C
Item 1	~~7 +~~	3 +	1 =	**$11.00**
Item 2	2 +	~~5 +~~	~~12 =~~	**$19.00**
Item 3	2 +	2 +	4 =	$8.00
Item 4	2 +	~~9 +~~	4 =	**$15.00**

The **value** keyword in the FormatColumn/FormatCell property indicates the value to be formatted.

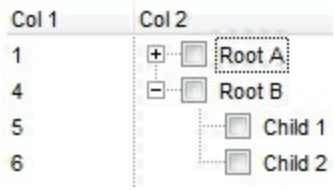
The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The Caption property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*

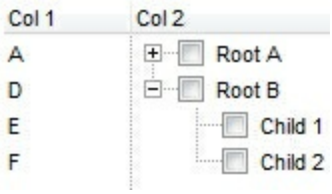
Other known operators for auto-numbering are:

- number **index** 'format', indicates the index of the item. The first added item has the index 0, the second added item has the index 1, and so on. The index of the item remains the same even if the order of the items is changed by sorting. For instance, 1 index '' gets the index of the item starting from 1 while 100 index '' gets the index of the item starting from 100. The number indicates the starting index, while the format is a set of characters to be used for specifying the index. If the format is missing, the index of the item is formatted as numbers. For instance: 1 index 'A-Z' gets the index as A, B, C... Z, BA, BB, ... BZ, CA, ... The 1 index 'abc' gives the index as: a,b,c,ba,bb,bc,ca,cb,cc,.... You can use other number formatting function to format the returned value. For instance "1 index '' format '0||2|:'" gets the numbers grouped by 2 digits and separated by : character.

  In the following screen shot the FormatColumn("Col 1") = "1 index ""

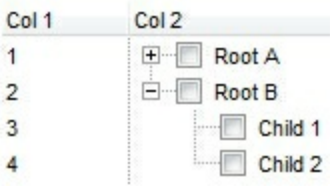


  In the following screen shot the FormatColumn("Col 1") = "1 index 'A-Z'"

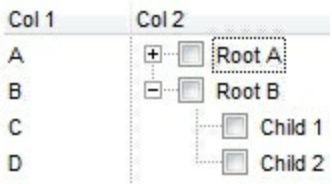


- number **apos** 'format' indicates the absolute position of the item. The first displayed item has the absolute position 0 (scrolling position on top), the next visible item is 1, and so on. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance, 1 apos '' gets the absolute position of the item starting from 1, while 100 apos '' gets the position of the item starting from 100. If the format is missing, the absolute position of the item is formatted as numbers.

  In the following screen shot the FormatColumn("Col 1") = "1 apos ""

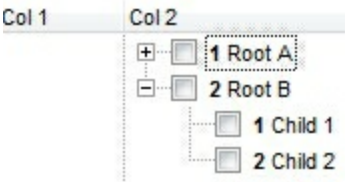


  In the following screen shot the FormatColumn("Col 1") = "1 apos 'A-Z'"

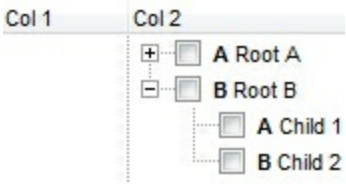


- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100. If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied, the opos gets the position of the item in the list of unfiltered items, while the pos gets the position of the item in the filtered list.*

  In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos '' + ' ' + value"

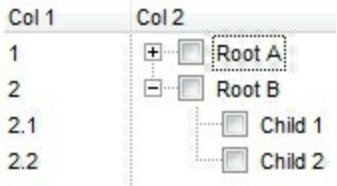


In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos 'A-Z' + ' ' + value"

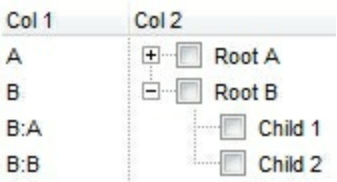


- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position.For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100. If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied, the opos gets the position of the item in the list of unfiltered items, while the pos gets the position of the item in the filtered list.*
- number **rpos** 'format' indicates the relative recursive position of the item. The recursive position indicates the position of the parent items too. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is of the following type "delimiter|format|format|...". If the format is missing, the delimiter is . character, and the positions are formatted as numbers. The format is applied consecutively to each parent item, from root to item itself.

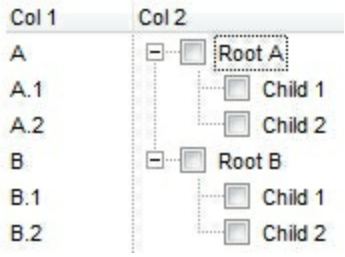
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ""



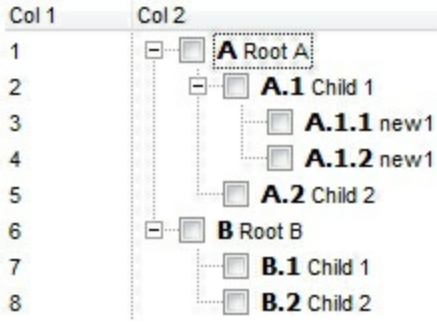
In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"



In the following screen shot the FormatColumn("Col 1") = "1 rpos '.|A-Z|'"



In the following screen shot the FormatColumn("Col 1") = "1 apos "" and FormatColumn("Col 2") = "'' + 1 rpos '.|A-Z|' + ' ' + value"



- number **rindex** 'format', number **rapos** 'format' and number **ropos** 'format' are working similar with number **rpos** 'format', excepts that they gives the index, absolute position, or the old child position.

This property/method supports predefined constants and operators/functions as described here.

property Items.IsItemVisible (Index as Long) as Boolean

Checks if the specific item is in the visible client area.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	A boolean expression that indicates whether the item fits the client area

Calls the EnsureVisibleItem method to ensure that an item fits the control's client area. Use the FirstVisibleItem, NextVisibleItem and IsItemVisible properties to get the items that fits the client area. The NextVisibleItem property gets the next visible item. Use the IsVisibleItem property to check whether an item fits the control's client area.

property Items.ItemAllowSizing(Index as Long) as Boolean

Retrieves or sets a value that indicates whether a user can resize the item at run-time.

Type	Description
Index as Long	A long expression that indicates the index of the item that can be resized.
Boolean	A Boolean expression that specifies whether the user can resize the item at run-time.

By default, the user can resize the item at run-time using mouse movements. Use the ItemAllowSizing property to specify whether a user can resize the item at run-time. Use the ItemsAllowSizing property to specify whether all items are resizable or not. Use the ItemHeight property to specify the height of the item. An item is resizable if the ItemAllowSizing property is True, or if the ItemsAllowSizing property is True (that means all items are resizable), and the ItemAllowSizing property is not False. For instance, if your application requires all items being resizable but only few of them being not resizable, you can have the ItemsAllowSizing property on True, and for those items that are not resizable, you can call the ItemAllowSizing property on False. The user can resize an item by moving the mouse between two items, so the vertical split cursor shows up, click and drag the mouse to the new position. Use the CellSingleLine property to specify whether the cell displays its caption using multiple lines. The ScrollBySingleLine property is automatically set on True, as soon as the user resizes an item.

property Items.ItemBackColor(Index as Long) as Color

Retrieves or sets a background color for a specific item.

Type	Description
Index as Long	A long expression that indicates the index of the item. If the Index is -1, the ItemBackColor changes the background color for all items.
Color	A color expression that indicates the item's foreground color. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the ItemBackColor property to change the item's background color. Use the CellForeColor property to change the cell's foreground color. Use the ItemForeColor property to change the item's foreground color. Use the CellBackColor property to change the cell's background color. Use the ClearItemBackColor property to clear the item's background color when the ItemBackColor property is used. You can use the ItemBackColor property and a skin (Add method) to define a different pattern on the item's background, when you need a special marker for the item.

In VB.NET or C# you require the following functions until the .NET framework will provide:

You can use the following VB.NET function:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
   Dim i As Long
i = c.R
i = i + 256 * c.G
i = i + 256 * 256 * c.B
ToUInt32 = Convert.ToUInt32(i)
End Function
```

You can use the following C# function:

```
private UInt32 ToUInt32(Color c)
{
   long i;
```

```
    i = c.R;
    i = i + 256 * c.G;
    i = i + 256 * 256 * c.B;
    return Convert.ToUInt32(i);
}
```

The following C# sample changes the background color for the focused item:

```
axList1.Items.set_ItemBackColor(axList1.Items.FocusItem, ToUInt32(Color.Red) );
```

The following VB.NET sample changes the background color for the focused item:

```
With AxList1.Items
   .ItemBackColor(.FocusItem) = ToUInt32(Color.Red)
End With
```

The following C++ sample changes the background color for the focused item:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetItemBackColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the background color for the focused item:

```
with thisform.List1.Items
   .ItemBackColor( .FocusItem ) = RGB(255,0,0)
endwith
```

property Items.ItemBold(Index as Long) as Boolean

Retrieves or sets a value that indicates whether the item should appear in bold.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	A boolean expression that specifies whether the item is bolded.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample bolds the selected item:

```
Dim iOldBold As Long

Private Sub List1_SelectionChanged()
   With List1.Items
      If Not (iOldBold = -1) Then
         .ItemBold(iOldBold) = False
      End If
      iOldBold = .SelectedItem()
      .ItemBold(iOldBold) = True
   End With
End Sub
```

The following VB sample bolds the focused item:

```
With List1.Items
   .ItemBold(.FocusItem) = True
End With
```

The following C++ sample bolds the focused item:

```
#include "Items.h"
CItems items = m_list.GetItems();
```

```
items.SetItemBold( items.GetFocusItem() , TRUE );
```

The following C# sample bolds the focused item:

```
axList1.Items.set_ItemBold(axList1.Items.FocusItem, true);
```

The following VB.NET sample bolds the focused item:

```
With AxList1.Items
   .ItemBold(.FocusItem) = True
End With
```

The following VFP sample bolds the focused item:

```
with thisform.List1.Items
   .ItemBold( .FocusItem ) = .t.
endwith
```

property Items.ItemBreak(Index as Long) as BreakLineEnum

Retrieves or sets a value that indicates whether the item is painted as a break line.

Type	Description
Index as Long	A long expression that indicates the index of the item.
BreakLineEnum	A BreakLineEnum expression that defines the type of break line.

The control draws the break line only if the cell's Caption is empty. Use the CellMerge property to combine two or more cells in a single cell.

The following sample adds a break line:

```
List1.Items.ItemBreak(List1.Items.Add()) = DoubleDotLine
```

property Items.ItemData(Index as Long) as Variant

Retrieves or sets a value that indicates the extra data for a specific item.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Variant	A VARIANT expression that indicates the item's extra data

Use the ItemData property to assign an extra value to an item. Use CellData property to associate an extra data with a cell. The ItemData and CellData are of Variant type, so you will be able to save here what ever you want: numbers, objects, strings, and so on. The user data is only for user use. The control doesn't use this value. Use the Data property to assign an extra data to a column. For instance, you can use the RemoveItem event to release any extra data that is associated to the item. Use the FindItemData property to search for item's data.

property Items.ItemFont (Index as Long) as IFontDisp

Retrieves or sets the item's font.

Type	Description
Index as Long	A long expression that specifies the index of item.
IFontDisp	A Font object being used for specified item.

By default, the ItemFont property is nothing. If the ItemFont property is nothing, the item uses the control's font. Use the ItemFont property to define a different font for the item. Use the CellFont and ItemFont properties to specify different fonts for cells or items. Use the CellBold, CellItalic, CellUnderline, CellStrikeout, ItemBold, ItemUnderline, ItemStrikeout, ItemItalic or CaptionFormat to specify different font attributes. Use the ItemHeight property to specify the height of the item. Use the Refresh method to refresh the control's content on the fly. Use the BeginUpdate and EndUpdate methods if you are doing multiple changes, so no need for an update each time a change is done.

The following VB sample changes the font for the focused item:

```
List1.BeginUpdate
With List1.Items
   .ItemFont(.FocusItem) = List1.Font
   With .ItemFont(.FocusItem)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
List1.EndUpdate
```

The following C++ sample changes the font for the focused item:

```
#include "Items.h"
#include "Font.h"
m_list.BeginUpdate();
CItems items = m_list.GetItems();
items.SetItemFont( items.GetFocusItem(), m_list.GetFont().m_lpDispatch );
COleFont font = items.GetItemFont( items.GetFocusItem() );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_list.EndUpdate();
```

The following VB.NET sample changes the font for the focused item:

```
AxList1.BeginUpdate()
With AxList1.Items
   .ItemFont(.FocusItem) = IFDH.GetIFontDisp(AxList1.Font)
   With .ItemFont(.FocusItem)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
AxList1.EndUpdate()
```

where the IFDH class is defined like follows:

```
Public Class IFDH
   Inherits System.Windows.Forms.AxHost

   Sub New()
      MyBase.New("")
   End Sub

   Public Shared Function GetIFontDisp(ByVal font As Font) As Object
      GetIFontDisp = AxHost.GetIFontFromFont(font)
   End Function

End Class
```

The following C# sample changes the font for the focused item:

```
axList1.BeginUpdate();
axList1.Items.set_ItemFont(axList1.Items.FocusItem, IFDH.GetIFontDisp(axList1.Font));
stdole.IFontDisp spFont = axList1.Items.get_ItemFont(axList1.Items.FocusItem);
spFont.Name = "Comic Sans MS";
spFont.Bold = true;
axList1.EndUpdate();
```

where the IFDH class is defined like follows:

```
internal class IFDH : System.Windows.Forms.AxHost
```

```
{
    public IFDH() : base("")
    {
    }

    public static stdole.IFontDisp GetIFontDisp(System.Drawing.Font font)
    {
        return System.Windows.Forms.AxHost.GetIFontFromFont(font) as stdole.IFontDisp;
    }
}
```

The following VFP sample changes the font for the focused item:

```
thisform.List1.Object.BeginUpdate()
with thisform.List1.Items
    .ItemFont(.FocusItem) = thisform.List1.Font
    with .ItemFont(.FocusItem)
        .Name = "Comic Sans MS"
        .Bold = .t.
    endwith
endwith
thisform.List1.Object.EndUpdate()
```

property Items.ItemForeColor(Index as Long) as Color

Retrieves or sets a foreground color for a specific item.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Color	A color expression that indicates the item's foreground color

Use the ItemForeColor property to change the item's foreground color. Use the CellForeColor property to change the cell's foreground color. Use the ItemBackColor property to change the item's background color. Use the ClearItemForeColor property to clear the item's foreground color once that the ItemForeColor property is used. Use the ForeColor property to specify the control's foreground color.

The following VB sample changes the foreground color of the focused item:

```
With List1.Items
  .ItemForeColor(.FocusItem) = vbRed
End With
```

In VB.NET or C# you require the following functions until the .NET framework will provide:

You can use the following VB.NET function:

```
  Shared Function ToUInt32(ByVal c As Color) As UInt32
    Dim i As Long
  i = c.R
  i = i + 256 * c.G
  i = i + 256 * 256 * c.B
  ToUInt32 = Convert.ToUInt32(i)
  End Function
```

You can use the following C# function:

```
private UInt32 ToUInt32(Color c)
{
  long i;
  i = c.R;
  i = i + 256 * c.G;
  i = i + 256 * 256 * c.B;
```

```
    return Convert.ToUInt32(i);
}
```

The following C# sample changes the foreground color of the focused item:

```
axList1.Items.set_ItemForeColor(axList1.Items.FocusItem, ToUInt32(Color.Red) );
```

The following VB.NET sample changes the foreground color of the focused item:

```
With AxList1.Items
   .ItemForeColor(.FocusItem) = ToUInt32(Color.Red)
End With
```

The following C++ sample changes the foreground color of the focused item:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetItemForeColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the foreground color of the focused item:

```
with thisform.List1.Items
   .ItemForeColor( .FocusItem ) = RGB(255,0,0)
endwith
```

property Items.ItemHeight(Index as Long) as Long

Retrieves or sets the item's height.

Type	Description
Index as Long	If the Index is -1, setting the ItemHeight property changes the height for all items. For instance, the ItemHeight(-1) = 24, changes the height for all items to be 24 pixels wide.
Long	A long expression that indicates the item's height in pixels.

Use the ItemHeight property to change the item's height. To change the default height of the item before inserting items to collection you can call DefaultItemHeight property of the control. The ExList control supports items with different heights. if the CellSingleLine property is False. Use the ItemHeight property to specify the height of the item when it contains no cells with CellSingleLine property on False. Use the ItemMaxHeight property to specify the maximum height of the item when it contains cells with CellSingleLine property on False. Use the ScrollBySingleLine property when using items with different heights. Use the ItemAllowSizing property to specify whether the user can resize the item at runtime.

property Items.ItemItalic(Index as Long) as Boolean

Retrieves or sets a value that indicates whether the item should appear in italic.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	Retrieves or sets a value that indicates whether the item should appear in italic.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample makes italic the focused item:

```
With List1.Items
   .ItemItalic(.FocusItem) = True
End With
```

The following C++ sample makes italic the focused item:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetItemItalic( items.GetFocusItem() , TRUE );
```

The following C# sample makes italic the focused item:

```
axList1.Items.set_ItemItalic(axList1.Items.FocusItem, true);
```

The following VB.NET sample makes italic the focused item:

```
With AxList1.Items
   .ItemItalic(.FocusItem) = True
End With
```

The following VFP sample makes italic the focused item:

```
with thisform.List1.Items
```

```
    .ItemItalic( .FocusItem ) = .t.
endwith
```

property Items.ItemMaxHeight(Index as Long) as Long

Retrieves or sets a value that indicates the maximum height when the item's height is variable.

Type	Description
Index as Long	A long expression that indicates the index of the item. For instance, the ItemMaxHeight(-1) = 24, changes the maximum-height for all items to be 24 pixels wide.
Long	A long value that indicates the maximum height when the item's height is variable.

By default, the ItemMaxHeight property is -1. The ItemMaxHeight property has effect only if it is greater than 0, and the item contains cells with CellSingleLine property on False. The ItemMinHeight property specifies the minimal height of the item while resizing. The CellSingleLine property specifies whether a cell displays its caption using multiple lines. The CellSingleLine property specifies whether a cell displays its caption using multiple lines. The ItemHeight property has no effect, if the CellSingleLine property is False. If the CellSingleLine property is False, you can specify the maximum height for the item using the ItemMaxHeight property. Use the ItemAllowSizing property to specify whether the user can resize the item at runtime.

property Items.ItemMinHeight(Index as Long) as Long

Retrieves or sets a value that indicates the minimum height when the item's height is sizing.

Type	Description
Index as Long	A long expression that indicates the index of the item. For instance, the ItemMinHeight(-1) = 24, changes the minimum-height for all items to be 24 pixels wide.
Long	A long value that indicates the minimum height when the item's height is variable.

By default, the ItemMinHeight property is -1. The ItemMinHeight property has effect only if the item contains cells with CellSingleLine property on False. The ItemMaxHeight property specifies the maximum height of the item while resizing. The CellSingleLine property specifies whether a cell displays its caption using multiple lines. The ItemHeight property has no effect, if the CellSingleLine property is False. If the CellSingleLine property is False, you can specify the minimum height for the item using the ItemMinHeight property. Use the ItemAllowSizing property to specify whether the user can resize the item at runtime

property Items.ItemPosition(Index as Long) as Long

Retrieves or sets a value that indicates the item's position.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Long	A long expression that indicates the item's position.

Use the ItemPosition property to change the item's position. The Sort method reorders the items, so each item position is changed. The item's position is zero based. Use the SortOrder property to sort a column. Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items as they are listed. The LastVisibleItem property retrieves the index of last item that fits the client area. The ItemPosition property is not available if the control is running in the virtual mode. Use the BackColorAlternate property to specify an alternate background color for odd an even items. You can use the GetItems(1) method to get the list of indexes for the items as they are displayed, sorted and filtered.

property Items.ItemStrikeOut(Index as Long) as Boolean

Retrieves or sets a value that indicates whether the item should appear in strikeout.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	A boolean expression that indicates whether the item is displayed with a horizontal line through it.

If the ItemStrikeOut property is True, the cell's font is displayed with a horizontal line through it. Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample draws a horizontal line through the focused item:

```
With List1.Items
   .ItemStrikeOut(.FocusItem) = True
End With
```

The following C++ sample draws a horizontal line through the focused item:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetItemStrikeOut( items.GetFocusItem() , TRUE );
```

The following C# sample draws a horizontal line through the focused item:

```
axList1.Items.set_ItemStrikeOut(axList1.Items.FocusItem, true);
```

The following VB.NET sample draws a horizontal line through the focused item:

```
With AxList1.Items
   .ItemStrikeOut(.FocusItem) = True
End With
```

The following VFP sample draws a horizontal line through the focused item:

```
with thisform.List1.Items
```

```
    .ItemStrikeOut( .FocusItem ) = .t.
endwith
```

property Items.ItemToVirtual (Index as Long) as Long

Gets the index of the virtual item giving the index of the item in the list.

Type	Description
Index as Long	A long expression that the index of the item in the list.
Long	A long expression that indicates the index of virtual item.

The ItemToVirtual property converts the index of the item in the list to the index of the virtual item/record. The ItemToVirtual property has effect only if the control is running in the virtual mode. Use the VirtualToItem property to get the index of the item in the list giving the index of the virtual item/record.

The following VB sample notifies the adoVirtual object that the user changes the date in the control:

```
Private Sub List1_AfterCellEdit(ByVal Index As Long, ByVal ColIndex As Long, ByVal
NewCaption As String)
   With List1.Items
      adoVirtual.Change .ItemToVirtual(Index), ColIndex, NewCaption
   End With
End Sub
```

property Items.ItemUnderline(Index as Long) as Boolean

Retrieves or sets a value that indicates whether the item is underlined.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	A boolean expression that indicates whether the item is underlined.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CaptionFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample underlines the focused item:

```
With List1.Items
   .ItemUnderline(.FocusItem) = True
End With
```

The following C++ sample underlines the focused item:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetItemUnderline( items.GetFocusItem() , TRUE );
```

The following C# sample underlines the focused item:

```
axList1.Items.set_ItemUnderline(axList1.Items.FocusItem, true);
```

The following VB.NET sample underlines the focused item:

```
With AxList1.Items
   .ItemUnderline(.FocusItem) = True
End With
```

The following VFP sample underlines the focused item:

```
with thisform.List1.Items
```

```
    .ItemUnderline( .FocusItem ) = .t.
endwith
```

property Items.LastVisibleItem ([Partially as Variant]) as Long

Retrieves the index of the last visible item.

Type	Description
Partially as Variant	A boolean expression that indicates whether the item is partially visible.
Long	A long expression that indicates the index of the last visible item.

The LastVisibleItem property retrieves the index of last item that fits the client area. Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items that fit the client area.

property Items.MatchItemCount as Long

Retrieves the number of items that match the filter.

Type	Description
Long	A long expression that specifies the number of matching items in the control. The value could be a positive value if no filter is applied, or negative while filter is on.

The MatchItemCount property counts the number of items that matches the current filter criteria. At runtime, the MatchItemCount property is a positive integer if no filter is applied, and negative if a filter is applied. If positive, it indicates the number of items within the control (Count property). If negative, a filter is applied, and the absolute value minus one, indicates the number of matching items after filter is applied.

The MatchItemCount property returns a value as explained bellow:

- 0, the control displays/contains no items, and no filter is applied to any column
- -1, the control displays no items, and there is a filter applied (no match found)
- positive number, indicates the number of items within the control (Count property)
- negative number, the absolute value minus 1, indicates the number of items that matches the current filter (match found)

property Items.NextVisibleItem (Index as Long) as Long

Retrieves the index of next visible item.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Long	A long expression that indicates the next visible item's index

The NextVisibleItem property retrieves -1 if there is no next visible item. Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items as they are listed. The LastVisibleItem property retrieves the index of last item that fits the client area. Use the PrevVisibleItem property to get the previous item. Use the SortOrder property to sort a column. Use the ItemPosition property to change the item's position. Use the ItemFromPoint property to get the item from cursor. Use the GetItems property to get the list of items.

The following VB sample displays the items as they are listed:

```
With List1.Items
   Dim i As Long
   i = .FirstVisibleItem
   While (i >= 0)
      Debug.Print .Caption(i, 0)
      i = .NextVisibleItem(i)
   Wend
End With
```

The following C++ sample displays the items as they are listed:

```
CItems items = m_list.GetItems();
long i = items.GetFirstVisibleItem();
while ( i >= 0 )
{
   CString strCaption = V2S( &items.GetCaption( i, COleVariant( long(0) ) ) );
   OutputDebugString( strCaption );
   i = items.GetNextVisibleItem( i );
}
```

The following VB.NET sample displays the items as they are listed:

```
With AxList1.Items
   Dim i As Integer = .FirstVisibleItem
   While (i >= 0)
      Debug.WriteLine(.Caption(i, 0))
      i = .NextVisibleItem(i)
   End While
End With
```

The following C# sample displays the items as they are listed:

```
int i = axList1.Items.FirstVisibleItem;
while (i >= 0)
{
   object cell = axList1.Items.get_Caption(i, 0);
   System.Diagnostics.Debug.WriteLine(cell != null ? cell.ToString() : "");
   i = axList1.Items.get_NextVisibleItem(i);
}
```

The following VFP sample displays the items as they are listed:

```
With thisform.List1.Items
   local i
   i = .FirstVisibleItem
   do While (i >= 0)
      wait window .Caption(i, 0)
      i = .NextVisibleItem(i)
   enddo
EndWith
```

property Items.PrevVisibleItem (Index as Long) as Long

Retrieves the index of previous visible item.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Long	A long expression that indicates the index of previous visible item.

The PrevVisibleItem property retrieves -1 if there is no previous visible item. Use the PrevVisibleItem property to get the previous visible item. Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items as they are listed. Use the SortOrder property to sort a column. Use the ItemPosition property to change the item's position. Use the ItemFromPoint property to get the item from cursor. Use the GetItems property to get the list of items.

method Items.Remove (Index as Long)

Removes a specific item.

Type	Description
Index as Long	A long expression that indicates the index of the item.

Use the Remove method to remove a specific item. The Remove method updates the indexes of the items in the list. For instance, if you remove the item 0, the items 1, 2, 3, ... becomes the 0,1,2,... Use the RemoveAll method to clear the Items collection. The control fires the RemoveItem event before removing an item. Use the RemoveItem event to release any extra data associated to the item. The Remove method is not available if the control is running in the virtual mode. Use the BeginUpdate and EndUpdate methods to maintain performance while removing items. Use the CellState property to specify whether an item is checked or unchecked. Use the SelectedItem property to retrieve the index of selected item(s). The RemoveSelection method removes the selected items

The following VB sample removes all selected items:

```
With List1
  .BeginUpdate
  With .Items
    While .SelectCount > 0
      .Remove .SelectedItem(0)
    Wend
  End With
  .EndUpdate
End With
```

The following VB sample removes the checked items:

```
With List1
  .BeginUpdate
  With .Items
    Dim i As Long
    For i = .Count - 1 To 0 Step -1
      If .CellState(i, 0) Then
        .Remove i
      End If
    Next
```

```
    End With
    .EndUpdate
End With
```

or you can use a sample like follows:

```
With List1
    .BeginUpdate
    With .Items
        Dim i As Long
        i = 0
        While i < .Count
            If (.CellState(i, 0)) Then
                .Remove (i)
            Else
                i = i + 1
            End If
        Wend
    End With
    .EndUpdate
End With
```

The following C++ sample removes the selected items:

```
#include "Items.h"

m_list.BeginUpdate();
CItems items = m_list.GetItems();
while ( items.GetSelectCount() )
    items.Remove(items.GetSelectedItem(0));
m_list.EndUpdate();
```

The following VFP sample removes the selected items:

```
with thisform.List1
    .BeginUpdate()
    with .Items
        do while ( .SelectCount() > 0 )
            .Remove(.SelectedItem(0))
```

```
        enddo
    endwith
    .EndUpdate()
endwith
```

method Items.RemoveAll ()

Removes all items from the control.

Type	Description

Use the RemoveAll items in order to clear the Items collection. The RemoveAll method doesn't clear the Columns collection. Use the Clear method to clear the remove all columns and all items. Use the Remove method to remove an item. The control fires the RemoveItem event before removing an item. The RemoveAll method is not available if the control is running in the virtual mode.

method Items.RemoveSelection ()

Removes the selected items.

Type	Description

The RemoveSelection method removes the selected items. The Remove method removes a specific item. The UnselectAll method unselects all items in the list.

property Items.SelectableItem(Index as Long) as Boolean

Specifies whether the user can select the item.

Type	Description
Index as Long	A long expression that indicates the index of the item being selectable.
Boolean	A boolean expression that specifies whether the item is selectable.

By default, all items are selectable. A selectable item is an item that user can select using the keys or the mouse. The SelectableItem property specifies whether the user can select an item. The SelectableItem property doesn't change the item's appearance. Use the ItemBreak property to add a break item. Use the ItemForeColor property to specify the item's foreground color. Use the ItemBackColor property to specify the item's background color. Use the ItemFont, ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to assign a different font to the item. Use the EnableItem property to disable an item. A disabled item looks grayed, but it is selectable. For instance, the user can't change the check box state in a disabled item. Use the SelectItem property to select an item. The ItemFromPoint property gets the item from point. For instance, if the user clicks a non selectable item the SelectionChanged event is not fired. A non selectable item is not focusable as well. It means that if the incremental searching is on, the non selectable items are ignored. Use the SelectCount property to get the number of selected items. Use the SelForeColor and SelBackColor properties to customize the colors for selected items.

The following VB sample makes not selectable the first visible item:

```
With List1.Items
   .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C++ sample makes not selectable the first visible item:

```
#include "Items.h"
CItems items = m_list.GetItems();
items.SetSelectableItem( items.GetFirstVisibleItem(), FALSE );
```

The following VB.NET sample makes not selectable the first visible item:

```
With AxList1.Items
   .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C# sample makes not selectable the first visible item:

```
axList1.Items.set_SelectableItem(axList1.Items.FirstVisibleItem, false);
```

The following VFP sample makes not selectable the first visible item:

```
with thisform.List1.Items
    .SelectableItem(.FirstVisibleItem) = .f.
endwith
```

method Items.SelectAll ()

Selects all items.

Type	Description

Use the SelectAll method to select all items in the list. The SelectAll method has effect only if the SingleSel property is False, if the control supports multiple items selection. Use the UnselectAll method to unselect all items in the list. Use the SelectItem property to select or unselect a specified item. Use the SelectedItem property to retrieve a value that indicates whether the item is selected or unselected. Use the SelectCount property to retrieve the number of selected items.

property Items.SelectCount as Long

Retrieves the count of selected items.

Type	Description
Long	A long expression that indicates the count of the selected items.

The control supports single or multiple selection. Use the SelectCount and SelectedItem properties to enumerate the collection of selected items. Use the SelectItem property to select or unselects programmatically an item. The SelectionChanged event is fired when the user changes the selection. Use the SelForeColor and SelBackColor properties to specify colors for selected items. Use the SingleSel property of the control to allow multiple selection. If the control supports only single selection (SingleSel property is True), the FocusItem retrieves the selected item too. Use the Caption property to specify the cell's caption. Use the SelectAll method to select all items in the list. Use the UnselectAll method to unselect all items in the list.

The following VB sample enumerates the collection of selected items:

```
Dim i As Long
With List1.Items
   For i = 0 To .SelectCount() - 1
      Debug.Print .Caption(.SelectedItem(i), 0)
   Next
End With
```

The following C++ sample enumerates the collection of selected items:

```
CItems items = m_list.GetItems();
for ( long i = 0; i < items.GetSelectCount(); i++ )
{
   CString strCaption = V2S( &items.GetCaption( items.GetSelectedItem(i), COleVariant(
long(0) ) ) );
   OutputDebugString( strCaption );
}
```

The following VB.NET sample enumerates the collection of selected items:

```
With AxList1.Items
   Dim i As Integer
```

```
    For i = 0 To .SelectCount() - 1
        Debug.WriteLine(.Caption(.SelectedItem(i), 0))
    Next
End With
```

The following C# sample enumerates the collection of selected items:

```
for ( int i = 0; i < axList1.Items.SelectCount; i++ )
{
    object cell = axList1.Items.get_Caption(axList1.Items.get_SelectedItem(i), 0);
    System.Diagnostics.Debug.WriteLine(cell != null ? cell.ToString() : "");
}
```

The following VFP sample enumerates the collection of selected items:

```
local i
With thisform.List1.Items
    For i = 0 To .SelectCount() - 1
        wait window nowait .Caption(.SelectedItem(i), 0)
    Next
EndWith
```

property Items.SelectedItem ([Index as Long]) as Long

Retrieves the selected item's index given its index into selected items collection.

Type	Description
Index as Long	A long expression that indicates the index (into the selected items collection) of selected item being accessed.
Long	A long expression that indicates the index of the selected item.

The control supports single or multiple selection, depends on SingleSel property. Use the SelectCount and SelectedItem properties to enumerate the collection of selected items. Use the SelectItem property to select or unselect programmatically an item. The SelectionChanged event is fired when the user changes the selection. Use the SelForeColor and SelBackColor properties to specify colors for selected items. If the control supports only single selection (SingleSel property is True), the FocusItem retrieves the selected item too. Use the Caption property to specify the cell's caption.

The following VB sample enumerates the collection of selected items:

```
Dim i As Long
With List1.Items
   For i = 0 To .SelectCount() - 1
      Debug.Print .Caption(.SelectedItem(i), 0)
   Next
End With
```

The following C++ sample enumerates the collection of selected items:

```
CItems items = m_list.GetItems();
for ( long i = 0; i < items.GetSelectCount(); i++ )
{
   CString strCaption = V2S( &items.GetCaption( items.GetSelectedItem(i), COleVariant(
long(0) ) ) );
   OutputDebugString( strCaption );
}
```

The following VB.NET sample enumerates the collection of selected items:

```
With AxList1.Items
```

```
    Dim i As Integer
    For i = 0 To .SelectCount() - 1
        Debug.WriteLine(.Caption(.SelectedItem(i), 0))
    Next
End With
```

The following C# sample enumerates the collection of selected items:

```
for ( int i = 0; i < axList1.Items.SelectCount; i++ )
{
    object cell = axList1.Items.get_Caption(axList1.Items.get_SelectedItem(i), 0);
    System.Diagnostics.Debug.WriteLine(cell != null ? cell.ToString() : "");
}
```

The following VFP sample enumerates the collection of selected items:

```
local i
With thisform.List1.Items
    For i = 0 To .SelectCount() - 1
        wait window nowait .Caption(.SelectedItem(i), 0)
    Next
EndWith
```

property Items.SelectItem(Index as Long) as Boolean

Selects or unselects a specific item.

Type	Description
Index as Long	A long expression that indicates the index of the item.
Boolean	A boolean expression that indicates whether the item is selected or unselected.

The control supports single or multiple selection. Use the SelectItem property to select or unselect programmatically an item. Use the SelectCount and SelectedItem properties to enumerate the collection of selected items. The SelectionChanged event is fired when the user changes the selection. Use the SelForeColor and SelBackColor properties to specify colors for selected items. Use the SingleSel property of the control to allow multiple selection. If the control supports only single selection (SingleSel property is True), the FocusItem retrieves the selected item too. Use the Caption property to specify the cell's caption. Use the SelectAll method to select all items in the list. Use the UnselectAll method to unselect all items in the list.

The following VB sample selects the first visible item:

```
With List1.Items
   .SelectItem(.FirstVisibleItem) = True
End With
```

The following VB sample selects all items in the list:

```
With List1
   .BeginUpdate
   With .Items
      For i = 0 To .Count - 1
         .SelectItem(i) = True
      Next
   End With
   .EndUpdate
End With
```

The following C++ sample selects the first visible item:

```
#include "Items.h"
CItems items = m_list.GetItems();
```

```
items.SetSelectItem( items.GetFirstVisibleItem(), TRUE );
```

The following VB.NET sample selects the first visible item:

```
With AxList1.Items
    .SelectItem(.FirstVisibleItem) = True
End With
```

The following C# sample selects the first visible item:

```
axList1.Items.set_SelectItem(axList1.Items.FirstVisibleItem, true);
```

The following VFP sample selects the first visible item:

```
with thisform.List1.Items
    .SelectItem(.FirstVisibleItem) = .t.
endwith
```

method Items.Sort (ColIndex as Variant, Ascending as Boolean)

Sorts a column.

Type	Description
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
Ascending as Boolean	A boolean expression that indicates whether the items are sorted ascending or descending

The Sort method uses the SortType property to determine the way how the items are ordered. Use the SortOrder property to sort a column and to display the sorting icon in the column's header. The Sort event is fired when the user sorts a column. The SortPosition property changes the position of the column in the control's sort bar. Use the DefaultSortOrder property to specify the column's default sort order, when the user first clicks the column's header. Use the ItemPosition property to change the item's position. Use the FirstVisibleItem and NextVisibleItem properties to enumerate the items as they are listed. The Sort method is not available if the control is running in the virtual mode.

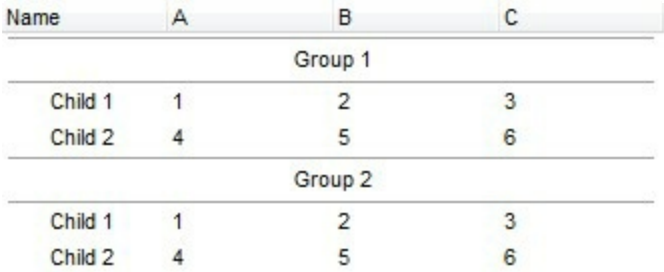
property Items.SortableItem(Index as Long) as Boolean

Specifies whether the item is sortable.

Type	Description
Index as Long	A long expression that indicates the index of the item being sortable.
Boolean	A boolean expression that specifies whether the item is sortable.

By default, all items are sortable. A sortable item can change its position after sorting. An unsortable item keeps its position after user performs a sort operation. Thought, the position of an unsortable item can be changed using the ItemPosition property. Use the SortableItem to specify a group item, a total item or a separator item. An unsortable item is not counted by a total field. The SortType property specifies the type of repositioning is being applied on the column when a sort operation is performed. The SortOrder property specifies whether the column is sorted ascendant or descendent. Use the Sort method to sort the items. Use the AllowSort property to avoid sorting a column when the user clicks the column. The SelectableItem property specifies whether an item can be selected.

The following screen shots shows the control when no column is sorted: (Group 1 and Group 2 has the SortableItem property on False)



The following screen shots shows the control when the column A is being sorted: (Group 1 and Group 2 keeps their original position after sorting)

method Items.UnselectAll ()

Unselects all items.

Type	Description

Use the UnselectAll method to unselect all items in the list. The UnselectAll method has effect only if the SingleSel property is False, if the control supports multiple items selection. Use the SelectAll method to select all items in the list. Use the SelectItem property to select or unselect a specified item. Use the SelectedItem property to retrieve a value that indicates whether the item is selected or unselected. Use the SelectCount property to retrieve the number of selected items. The RemoveSelection method removes the selected items.

property Items.VirtualToItem (Index as Long) as Long

Gets the index of the item in the list giving the index of the virtual item.

Type	Description
Index as Long	A long expression that indicates the index of the virtual item.
Long	A long expression that indicates the index of the item in the list.

The VirtualToItem property converts the the index of the virtual item/record to the index of the item in the list. The VirtualToItem property scrolls the control's content to make sure that the virtual item is in the control's client area. The VirtualToItem property has effect only if the control is running in the virtual mode. Use the ItemToVirtual property to get the index of the virtual item based on the index of the item in the list.

The following VB sample notifies the adoVirtual object that the user changes the date in the control:

```
Private Sub List1_AfterCellEdit(ByVal Index As Long, ByVal ColIndex As Long, ByVal
NewCaption As String)
  With List1.Items
    adoVirtual.Change .ItemToVirtual(Index), ColIndex, NewCaption
  End With
End Sub
```

property Items.VisibleCount as Long

Retrieves the number of visible items.

Type	Description
Long	A long expression that indicates the count of the visible items.

The VisibleCount property counts the number of the items that fits the control's client area (partially visible items are not counted). Use FirstVisibleItem and NextVisibleItem properties to determine the items that fit the client area. Use the IsItemVisible property to check whether an item fits the control's client area. Use the Count property to count the items in the control.

property Items.VisibleItemCount as Long

Retrieves the number of visible items.

Type	Description
Long	A long expression that specifies the number of visible items in the control. The value could be a positive value if no filter is applied, or negative while filter is on.

The VisibleItemCount property counts the number of visible items in the list. For instance, you can use the VisibleItemCount property to get the number the control displays once the user applies a filter.

The VisibleItemCount property returns a value as explained bellow:

- 0, the control displays/contains no items, and no filter is applied to any column
- -1, the control displays no items, and there is a filter applied (no match found)
- positive number, indicates the number of visible items, and the control has no filter applied to any column
- negative number, the absolute value munus 1, indicates the number of visible items, and there is a filter applied (match found)

The VisibleCount property retrieves the number of items being displayed in the control's client area. Use FirstVisibleItem and NextVisibleItem properties to determine the items being displayed in the control's client area. Use the IsItemVisible property to check whether an item fits the control's client area. Use the Count property to count the items in the control.

List 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: {1B0CA5A8-2107-4460-BBEE-F25F8801B2F6}. The object's program identifier is: "Exontrol.List". The /COM object module is: "ExList.dll"

Add an advanced List control to your application. The exList ActiveX control is 32-bit light ActiveX, that displays and edit your tabular data. 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 List control supports the following properties and methods:.

Name	Description
AllowEdit	Retrieves or sets a value that indicates whether the editing list is allowed or disabled.
AllowSelectNothing	Specifies whether the current selection is erased, once the user clicks outside of the items section.
AnchorFromPoint	Retrieves the identifier of the anchor from point.
Appearance	Retrieves or sets the control's appearance.
ApplyFilter	Applies the filter.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoDrag	Gets or sets a value that indicates the way the component supports the AutoDrag feature.
AutoSearch	Enables or disables incremental search feature.
BackColor	Retrieves or sets a value that indicates the control's background color.
BackColorAlternate	Specifies the background color used to display alternate items in the control.
BackColorHeader	Specifies the header's background color.
BackColorLevelHeader	Specifies the multiple levels header's background color.
BackColorLock	Retrieves or sets a value that indicates the control's background color for the locked area.
BackColorSortBar	Retrieves or sets a value that indicates the sort bar's background color.
BackColorSortBarCaption	Returns or sets a value that indicates the caption's background color in the control's sort bar.
	Returns or sets a value that indicates the background

Background	color for parts in the control.
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.
CheckImage	Retrieves or sets a value that indicates the index of image used by cells of checkbox type.
ClearFilter	Clears the filter.
ColumnAutoResize	Returns or sets a value indicating whether the control will automatically size its visible columns to fit on the control's client width.
ColumnFromPoint	Retrieves the column from point.
Columns	Retrieves the control's column collection.
ColumnsAllowSizing	Retrieves or sets a value that indicates whether a user can resize columns at run-time.
ConditionalFormats	Retrieves the conditional formatting collection.
ContinueColumnScroll	Retrieves or sets a value indicating whether the control will automatically scroll the visible columns by pixel or by column width.
Copy	Copies the control's content to the clipboard, in the EMF format.
CopyTo	Exports the control's view to an EMF file.
CountLockedColumns	Retrieves or sets a value indicating the number of locked columns. A locked column is not scrollable.
DataSource	Retrieves or sets a value that indicates the data source for object.
DefaultItemHeight	Retrieves or sets a value that indicates the default item height.
Description	Changes descriptions for control objects.
DetectAddNew	Specifies whether the control detects when a new record is added to the bounded recordset.
DetectDelete	Specifies whether the control detects when a record is deleted from the bounded recordset.
DrawGridLines	Retrieves or sets a value that indicates whether the grid lines are visible or hidden.
Enabled	Enables or disables the control.

EndUpdate	Resumes painting the control after painting is suspended by the BeginUpdate method.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
Export	Exports the control's data to a CSV format.
FilterBarBackColor	Specifies the background color of the control's filter bar.
FilterBarCaption	Specifies the filter bar's caption.
FilterBarDropDownHeight	Specifies the height of the drop down filter window proportionally with the height of the control's list.
FilterBarFont	Retrieves or sets the font for control's filter bar.
FilterBarForeColor	Specifies the foreground color of the control's filter bar.
FilterBarHeight	Specifies the height of the control's filter bar. If the value is less than 0, the filter bar is automatically resized to fit its description.
FilterBarPrompt	Specifies the caption to be displayed when the filter pattern is missing.
FilterBarPromptColumns	Specifies the list of columns to be used when filtering using the prompt.
FilterBarPromptPattern	Specifies the pattern for the filter prompt.
FilterBarPromptType	Specifies the type of the filter prompt.
FilterBarPromptVisible	Shows or hides the filter prompt.
FilterCriteria	Retrieves or sets the filter criteria.
Font	Retrieves or sets the control's font.
ForeColor	Retrieves or sets a value that indicates the control's foreground color.
ForeColorHeader	Specifies the header's foreground color.
ForeColorLock	Retrieves or sets a value that indicates the control's foreground color for the locked area.
ForeColorSortBar	Retrieves or sets a value that indicates the sort bar's foreground color.
FormatABC	Formats the A,B,C values based on the giving expression and returns the result.
FormatAnchor	Specifies the visual effect for anchor elements in HTML

	captions.
FreezeEvents	Prevents the control to fire any event.
FullRowSelect	Enables full-row selection in the control.
GetItems	Gets the collection of items into a safe array,
GridLineColor	Specifies the grid line color.
GridLineStyle	Specifies the style for gridlines in the list part of the control.
HeaderAppearance	Retrieves or sets a value that indicates the header's appearance.
HeaderHeight	Retrieves or sets a value indicating control's header height.
HeaderSingleLine	Specifies whether the control resizes the columns header and wraps the captions in single or multiple lines.
HeaderVisible	Retrieves or sets a value that indicates whether the the list's header is visible or hidden.
HideSelection	Returns a value that determines whether selected item appears highlighted when a control loses the focus.
HotBackColor	Retrieves or sets a value that indicates the hot-tracking background color.
HotForeColor	Retrieves or sets a value that indicates the hot-tracking foreground color.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
HyperLinkColor	Specifies the hyperlink color.
Images	Sets at runtime the control's image list. The Handle should be a handle to an Image List control (HIMAGELIST type).
ImageSize	Retrieves or sets the size of icons the control displays.
ItemFromPoint	Retrieves the item from point.
Items	Retrieves the control's item collection.
ItemsAllowSizing	Retrieves or sets a value that indicates whether a user can resize items at run-time.
Layout	Saves or loads the control's layout, such as positions of the columns, scroll position, filtering values.
MarkSearchColumn	Retrieves or sets a value that indicates whether the searching column is marked or unmarked

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
PictureDisplayLevelHeader	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's header background.
PictureLevelHeader	Retrieves or sets a graphic to be displayed in the control's header when multiple levels is on.
PutItems	Adds an array of integer, long, date, string, double, float, or variant arrays to the list, beginning at Index.
RadioImage	Retrieves or sets a value that indicates the index of image used by cells of radio type.
RClickSelect	Retrieves or sets a value that indicates whether an item is selected using right mouse button.
Refresh	Refreshes the control's content.
RemoveSelection	Removes the selected items (including the descendents)
ReplaceIcon	Adds a new icon, replaces an icon or clears the control's image list.
RightToLeft	Indicates whether the component should draw right-to-left for RTL languages.
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.
ScrollBySingleLine	Retrieves or sets a value that indicates whether the control scrolls the lines to the end. If you have at least a cell that has SingleLine false, you have to check the ScrollBySingleLine property.
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.
SearchColumnIndex	Retrieves or sets a value indicating the column's index that is used for auto search feature.
SelBackColor	Retrieves or sets a value that indicates the selection background color.
SelBackMode	Retrieves or sets a value that indicates whether the selection is transparent or opaque.
SelectColumnIndex	Retrieves or sets a value that indicates control column's index where the user is able to select an item. It has effect only for FullRowSelect = false.
SelectOnRelease	Indicates whether the selection occurs when the user releases the mouse button.
SelForeColor	Retrieves or sets a value that indicates the selection foreground color.
SelLength	Returns or sets the number of characters selected.
SelStart	Returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.
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.
ShowToolTip	Shows the specified tooltip at given position.
SingleSel	Retrieves or sets a value that indicates whether the control supports single or multiple selection.

SingleSort	Returns or sets a value that indicates whether the control supports sorting by single or multiple columns.
SortBarCaption	Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns.
SortBarColumnWidth	Specifies the maximum width a column can be in the control's sort bar.
SortBarHeight	Retrieves or sets a value that indicates the height of the control's sort bar.
SortBarVisible	Retrieves or sets a value that indicates whether control's sort bar is visible or hidden.
SortOnClick	Retrieves or sets a value that indicates whether the control sorts automatically the data when the user click on column's caption.
Statistics	Gives statistics data of objects being hold by the control.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
TemplatePut	Defines inside variables for the next Template/ExecuteTemplate call.
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.
UnboundHandler	Specifies the control's unbound handler.
UseTabKey	Retrieves or sets a value indicating whether the control uses tab key for changing the searching column.
UseVisualTheme	Specifies whether the control uses the current visual theme to display certain UI parts.
Version	Retrieves the control's version.
VirtualMode	Specifies a value that indicates whether the control is running in the virtual mode.
VisualAppearance	Retrieves the control's appearance.

VisualDesign	Invokes the control's VisualAppearance designer.

property List.AllowEdit as Boolean

Retrieves or sets a value that indicates whether the editing list is allowed or disabled.

Type	Description
Boolean	A boolean expression that indicates whether the editing list is allowed or disabled.

By default, the AllowEdit property is false. If the AllowEdit property is True, the control fires the BeforeCellEdit event just before editing a cell, and fires the AfterCellEdit after that edit operation ends. Use the Edit method to pragmatically edit an item. Use the SelStart and SelLenght properties to specify the selected text when edit operation starts.

property List.AllowSelectNothing as Boolean

Specifies whether the current selection is erased, once the user clicks outside of the items section.

Type	Description
Boolean	A Boolean expression that specifies whether the current selection is erased, once the user clicks outside of the items section.

By default, the AllowSelectNothing property is False. The AllowSelectNothing property specifies whether the current selection is erased, once the user clicks outside of the items section. For instance, if the control's SingleSel property is True, and AllowSelectNothing property is True, you can un-select the single-selected item if pressing the CTRL + Space, or by CTRL + click.

property List.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 List1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With List1
    .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 AxList1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_MouseMoveEvent) Handles AxList1.MouseMoveEvent
  With AxList1
    .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 axList1_MouseMoveEvent(object sender,
AxEXLISTLib._IListEvents_MouseMoveEvent e)
{
   axList1.ShowToolTip(axList1.get_AnchorFromPoint(-1, -1));
}
```

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

```
void OnMouseMoveList1(short Button, short Shift, long X, long Y)
{
   COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
   m_list.ShowToolTip( m_list.GetAnchorFromPoint( -1, -1 ), vtEmpty, vtEmpty, vtEmpty );
}
```

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

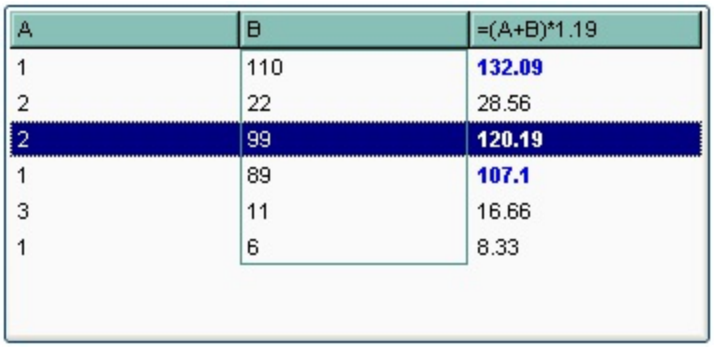
```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform
   With .List1
      .ShowToolTip(.AnchorFromPoint(-1, -1))
   EndWith
endwith
```

property List.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

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

Use the Appearance property to specify the control's border. Use the HeaderAppearance property to change the control's header bar appearance. Use the Add method to add new skins to the control. Use the BackColor property to specify the control's background color. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips.



A	B	=(A+B)*1.19
1	110	132.09
2	22	28.56
2	99	120.19
1	89	107.1
3	11	16.66
1	6	8.33

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

```
With List1
  .BeginUpdate
    .VisualAppearance.Add &H16, "c:\temp\frame.ebn"
    .Appearance = &H16000000
    .BackColor = RGB(250, 250, 250)
```

```
    .EndUpdate
End With
```

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

```
With AxList1
    .BeginUpdate()
    .VisualAppearance.Add(&H16, "c:\temp\frame.ebn")
    .Appearance = &H16000000
    .BackColor = Color.FromArgb(250, 250, 250)
    .EndUpdate()
End With
```

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

```
axList1.BeginUpdate();
axList1.VisualAppearance.Add(0x16, "c:\\temp\\frame.ebn");
axList1.Appearance = (EXLISTLib.AppearanceEnum)0x16000000;
axList1.BackColor = Color.FromArgb(250, 250, 250);
axList1.EndUpdate();
```

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

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

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

```
with thisform.List1
    .BeginUpdate
        .VisualAppearance.Add(0x16, "c:\temp\frame.ebn")
        .Appearance = 0x16000000
        .BackColor = RGB(250, 250, 250)
    .EndUpdate
endwith
```

method List.ApplyFilter ()

Applies the filter.

Type	Description

The ApplyFilter method updates the control's content once that user sets the filter using the Filter and FilterType properties. Use the ClearFilter method to clear the control's filter. Use the DisplayFilterButton property to show the filter drop down button in the column's caption. Use the FilterCriteria property to specify the filter criteria usinr OR, AND or NOT operators. Use the CustomFilter property to define you custom filters. Use the ShowFilter method to show programmatically the column's drop down filter window.

method List.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 List1_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.

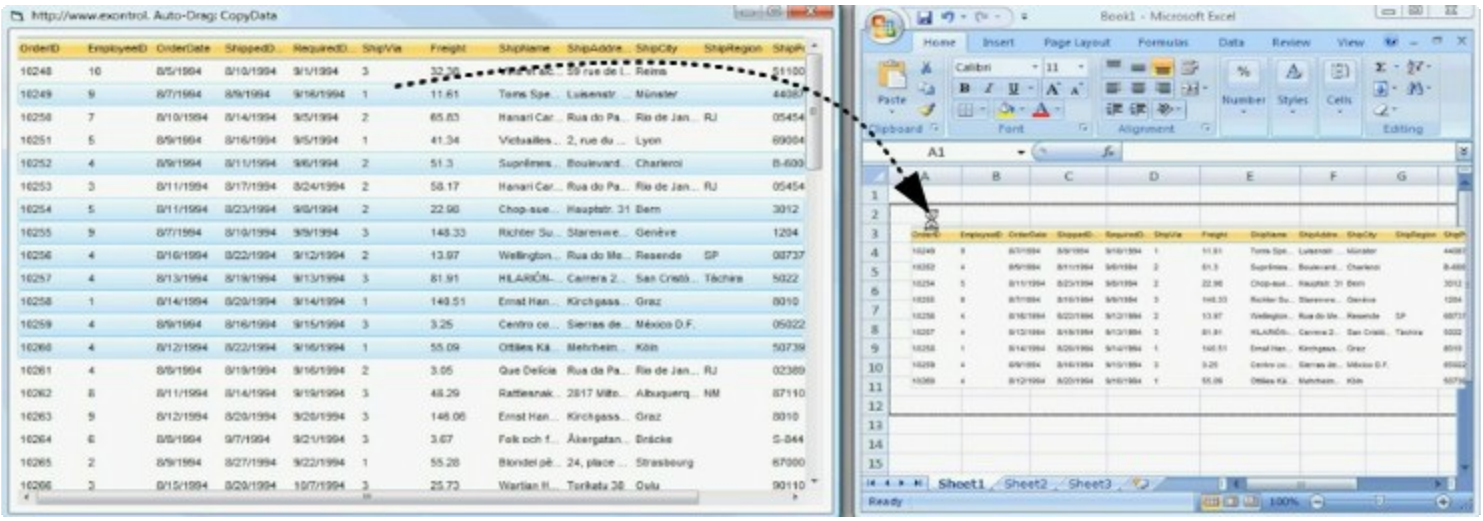
The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

property List.AutoDrag as AutoDragEnum

Gets or sets a value that indicates the way the component supports the AutoDrag feature.

Type	Description
AutoDragEnum	An AutoDragEnum expression that specifies what the control does once the user clicks and start dragging an item.

By default, the AutoDrag property is exAutoDragNone(0). The AutoDrag feature indicates what the control does when the user clicks an item and starts dragging it. For instance, using the AutoDrag feature you can automatically lets the user to drag and drop the data to OLE compliant applications like Microsoft Word, Excel and so on. The SingleSel property specifies whether the control supports single or multiple selection. The AutoDrag feature adds automatically Drag and Drop, but you can still use the OLEDropMode property to handle the OLE Drag and Drop event for your custom action. The control fires the AllowAutoDrag event, when the AutoDrag property is exAutoDragPosition.



The drag and drop operation starts:

- once the user clicks and moves the cursor up or down, if the SingleSel property is True.
- once the user clicks, and waits for a short period of time, if SingleSel property is False (multiple items in selection is allowed). In this case, you can drag and drop any item that is not selected, or a contiguously selection

Once the drag and drop operation starts the mouse pointer is changed to MOVE cursor if the operation is possible, else if the Drag and Drop operation fails or if it is not possible, the mouse pointer is changed to NO cursor.

If using the AutoDrag property on:

- exAutoDragPosition

the Drag and Drop starts only:

- item from cursor is a selectable (SelectableItem property on True, default) and sortable item (SortableItem property on True, default).
- if multiple items are selected, the selection is contiguously.

Use the AutoDrag property to allow Drag and Drop operations like follows:

- Ability to change the column or row position without having to manually add the OLE drag and drop events
- Ability to drag and drop the data as *text*, to your favorite Office applications, like Word, Excel, or any other OLE-Automation compliant
- Ability to drag and drop the data as it *looks*, to your favorite Office applications, like Word, Excel, or any other OLE-Automation compliant
- Ability to smoothly scroll the control's content moving the mouse cursor up or down
- and more ...

property List.AutoSearch as Boolean

Enables or disables incremental search feature.

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

By default, the AutoSearch property is True. The auto-search feature is is commonly known as incremental search. An incremental search begins searching as soon as you type the first character of the search string. As you type in the search string, the control selects the item (and highlight the portion of the string that match where the string (as you have typed it so far) would be found. The control supports 'starts with' or 'contains' incremental search as described in the AutoSearch property of the Column object. Use the MarkSearchColumn property to specify whether the control draws a rectangle around the searching column. The SearchColumnIndex property specifies the index of the column where incremental search feature works.

property List.BackColor as Color

Retrieves or sets a value that indicates the control's background color.

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

Use the BackColor property to change the control's background color. Use the ForeColor property to change the control's foreground color. Use the ItemBackColor or CellBackColor to change the item or cell's background color. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the Def(exCellBackColor) property to specify the background color for all cells in a column. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. To change the background color of the control's locked area use BackColorLock property. Use the CellBackColor property to assign a different background color for a specified cell. Use the ItemBackColor property to specify the item's background color. Use the BackColorAlternate property to specify the background color used to display alternate items in the control. Use the Picture property to assign a picture to the control's background.

property List.BackColorAlternate as Color

Specifies the background color used to display alternate items in the control.

Type	Description
Color	Specifies the background color used to display alternate items in the control

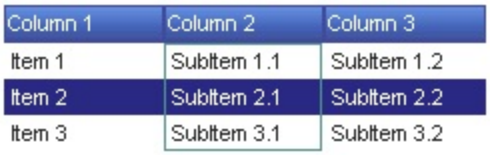
By default, the control's BackColorAlternate property is zero. The control ignores the BackColorAlternate property if it is 0 (zero). Use the BackColor property to specify the control's background color. Use the SelBackColor property to specify the selection background color. Use the ItemPosition property to change the item's position.

property List.BackColorHeader as Color

Specifies the header's background color.

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

Use the BackColorHeader and ForeColorHeader properties to define colors used to paint the control's header bar. Use the BackColorLevelHeader property to specify the background color of the control's header bar when multiple levels are displayed. Use the LevelKey property to display the control's header bar using multiple levels. If the control displays the header bar using multiple levels the HeaderHeight property gets the height in pixels of a single level in the header bar. The control's header displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the BackColorSortBar property to specify the background color of the control's sort bar.



The following VB sample changes the visual appearance for the control's header. 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 BackColorHeader property to indicates the index of the skin that we want to use. The sample applies the "" to the control' header bar:

```
With List1
   With .VisualAppearance
      .Add &H24, App.Path + "\header.ebn"
   End With
   .BackColorHeader = &H24000000
   .ForeColorHeader = RGB(255, 255, 255)
End With
```

The following C++ sample changes the visual aspect of the control' header bar:

```
#include "Appearance.h"
m_list.GetVisualAppearance().Add( 0x24,
COleVariant(_T("D:\\Temp\\ExList.Help\\header.ebn")) );
m_list.SetBackColorHeader( 0x24000000 );
m_list.SetForeColorHeader( RGB(255,255,255) );
```

The following VB.NET sample changes the visual aspect of the control' header bar:

```
With AxList1
   With .VisualAppearance
      .Add(&H24, "D:\Temp\ExList.Help\header.ebn")
   End With
   .Template = "BackColorHeader = 603979776"
   .ForeColorHeader = RGB(255,255,255)
End With
```

The 603979776 value indicates the &H24000000 in hexadecimal.

The following C# sample changes the visual aspect of the control' header bar:

```
axList1.VisualAppearance.Add(0x24, "D:\\Temp\\ExList.Help\\header.ebn");
axList1.Template = "BackColorHeader = 603979776";
axList1.ForeColorHeader = Color.White;
```

The 603979776 value indicates the 0x24000000 in hexadecimal.

The following VFP sample changes the visual aspect of the control' header bar:

```
With thisform.List1
   With .VisualAppearance
      .Add(36, "D:\Temp\ExList.Help\header.ebn")
   EndWith
   .BackColorHeader = 603979776
   .ForeColorHeader = RGB(255,255,255)
EndWith
```

property List.BackColorLevelHeader as Color

Specifies the multiple levels header's background color.

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

By default, the BackColorPropertyLevelHeader property is the same as the control's BackColorHeader property. Use the BackColorLevelHeader property to specify a background color for parts of the control's header that are not occupied by the column's headers. The BackColorLevelHeader property has effect only if there are two or more neighbor columns with the same non empty level key. Use the LevelKey property to specify the control's level key. The HeaderHeight property indicates the height in pixels of a single level in the control's header bar. Use the PictureLevelHeader property to specify a picture being displayed on the multiple levels header bar.

property List.BackColorLock as Color

Retrieves or sets a value that indicates the control's background color for the locked area.

Type	Description
Color	A color expression that indicates the color used to paint the control's background locked area.

The ExList control can group the control columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. Use ForeColorLock property to change the foreground color of the control's locked area. Use the Def(exCellBackColor) property to specify the background color for all cells in the column. Use the Def(exCellForeColor) property to specify the foreground color for all cells in the column.

property List.BackColorSortBar as Color

Retrieves or sets a value that indicates the sort bar's background color.

Type	Description
Color	A color expression that indicates the background color of the sort bar. 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.

Use the BackColorSortBar property to specify the background color of the control's sort bar. Use the SortBarVisible property to show the control's sort bar. Use the BackColorSortBarCaption property to specify the background color of the caption of the sort bar. The caption of the sort bar is visible, if there are no columns in the sort bar. Use the SortBarCaption property to specify the caption of the sort bar. Use the ForeColorSortBar property to specify the foreground color of the control's sort bar. Use the BackColor property to specify the control's background color. Use the BackColorHeader property to specify the background color of the control's header bar. Use the BackColorLevelHeader property to specify the background color of the control's header bar when multiple levels are displayed.

property List.BackColorSortBarCaption as Color

Returns or sets a value that indicates the caption's background color in the control's sort bar.

Type	Description
Color	A color expression that indicates the caption's background color in the control's sort bar. 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.

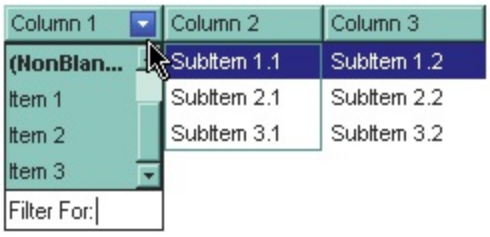
Use the SortBarCaption property to specify the caption of the sort bar, when the control's sort bar contains no columns. Use the BackColorSortBar property to specify the background color of the control's sort bar. Use the ForeColorSortBar property to specify the foreground color of the caption in the control's sort bar.

property List.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 "drop down" filter button. The sample applies the skin "" to the "drop down" filter buttons:

```
With List1
   With .VisualAppearance
      .Add &H1, App.Path + "\fbardd.ebn"
   End With
   .Background(exHeaderFilterBarButton) = &H1000000
End With
```

The following C++ sample changes the visual appearance for the "drop down" filter button:

```
#include "Appearance.h"
```

```
m_list.GetVisualAppearance().Add( 0x01,
COleVariant(_T("D:\\Temp\\ExList.Help\\fbardd.ebn")) );
m_list.SetBackground( 0 /*exHeaderFilterBarButton*/, 0x1000000 );
```

The following VB.NET sample changes the visual appearance for the "drop down" filter button:

```
With AxList1
   With .VisualAppearance
      .Add(&H1, "D:\Temp\ExList.Help\fbardd.ebn")
   End With
   .set_Background(EXLISTLib.BackgroundPartEnum.exHeaderFilterBarButton, &H1000000)
End With
```

The following C# sample changes the visual appearance for the "drop down" filter button:

```
axList1.VisualAppearance.Add(0x1, "D:\\Temp\\ExList.Help\\fbardd.ebn");
axList1.set_Background(EXLISTLib.BackgroundPartEnum.exHeaderFilterBarButton,
0x1000000);
```

The following VFP sample changes the visual appearance for the "drop down" filter button:

```
With thisform.List1
   With .VisualAppearance
      .Add(1, "D:\Temp\ExList.Help\fbardd.ebn")
   EndWith
   .Object.Background(0) = 16777216
EndWith
```

The 16777216 value is the 0x1000000 value in hexadecimal.

method List.BeginUpdate ()

Maintains performance when items are added to the control one at a time.

Type	Description

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

The following VB sample prevents painting the control while the control loads data from a recordset:

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Orders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Opens the table using static mode

With List1
   .BeginUpdate
   For Each f In rs.Fields
      .Columns.Add f.Name
   Next
   .PutItems rs.GetRows()
   .EndUpdate
End With
```

The following C++ sample prevents refreshing the control while adding columns and items from an ADODB recordset:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"

#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;

_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
```

```
{
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_list.BeginUpdate();
         m_list.SetColumnAutoResize( FALSE );
         CColumns columns = m_list.GetColumns();
         long nCount = spRecordset->Fields->Count;
         if ( nCount > 0 )
         {
            // Adds the columns
            for ( long i = 0 ; i < nCount; i++ )
               columns.Add( spRecordset->Fields->Item[ i ]->Name );
            m_list.PutItems( &spRecordset->GetRows(-1), vtMissing );
         }
         m_list.EndUpdate();
      }
   }
   catch ( _com_error& e )
   {
      AfxMessageBox( e.Description() );
   }
}
```

The sample adds a column for each field in the recordset, and add a new items for each record. You can use the DataSource property to bind a recordset to the control. The #import statement imports definitions for ADODB type library, that's used to fill the control.

The following VB.NET sample prevents refreshing the control while adding columns and items:

```
With AxList1
   .BeginUpdate()
   With .Columns
      .Add("Column 1")
      .Add("Column 2")
   End With
   With .Items
      Dim iNewItem As Integer
      iNewItem = .Add("Item 1")
      .Caption(iNewItem, 1) = "SubItem 1"
      iNewItem = .Add("Item 2")
      .Caption(iNewItem, 1) = "SubItem 2"
   End With
   .EndUpdate()
End With
```

The following C# sample prevents refreshing the control while adding columns and items:

```
axList1.BeginUpdate();
EXLISTLib.Columns columns = axList1.Columns;
columns.Add("Column 1");
columns.Add("Column 2");
EXLISTLib.Items items = axList1.Items;
int iNewItem = items.Add("Item 1");
items.set_Caption(iNewItem, 1, "SubItem 1");
iNewItem = items.Add("Item 2");
items.set_Caption(iNewItem, 1, "SubItem 2");
axList1.EndUpdate();
```

The following VFP sample prevents refreshing the control while adding new columns and items:

```
thisform.List1.BeginUpdate()
with thisform.List1.Columns
   .Add("Column 1")
   .Add("Column 2")
endwith
```

```
with thisform.List1.Items
   local i
   i = .Add("Item 1")
   .Caption(i, 1) = "SubItem 1"
   i = .Add("Item 2")
   .Caption(i, 1) = "SubItem 2"
endwith
thisform.List1.EndUpdate()
```

property List.CheckImage(State as CheckStateEnum) as Long

Retrieves or sets a value that indicates the index of image used by cells of checkbox type.

Type	Description
State as CheckStateEnum	A long expression that indicates the check' state: 0 - unchecked, 1 - checked, 2 - partial checked.
Long	A long expression that indicates the index image used for painting the cells of check type. The last 7 bits in the high significant byte of the long 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.

Use CheckImage and RadioImage properties to define icons for radio and check box cells. The CheckImage property defines the index of the icon being used by check boxes. Use the CellHasCheckBox property to assign a checkbox to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellImage or CellImages property to assign one or multiple icons to a cell. Use the CellPicture property to assign a picture to a cell. Use the CellStateChanged event to notify your application when the cell's state is changed. The ImageSize property defines the size (width/height) of the control's check-box/radio-button.

method List.ClearFilter ()

Clears the filter.

Type	Description

The method clears the Filter and FilterType properties for all columns in the control, excepts for exNumeric and exCheck values where only the Filter property is set on empty. The ApplyFilter method is automatically called when ClearFilter method is invoked. Use the FilterBarHeight property to hide the control's filter bar. Use the FilterBarCaption property to specify the caption in the control's filter bar. Use the Description property to change predefined strings in the control's filter bar. Use the Background property to change the visual appearance for the closing button in the control's filter bar. Use the CustomFilter property to define you custom filters. Use the ShowFilter method to show programmatically the column's drop down filter window.

property List.ColumnAutoResize as Boolean

Returns or sets a value indicating whether the control will automatically size its visible columns to fit on the control's client width.

Type	Description
Boolean	A boolean expression indicating whether the control will automatically size its visible columns to fit on the control's client width.

Use the ColumnAutoResize property to fit all your columns in the client area. Use the Width property to specify the column's width. Use the SortBarColumnWidth property to specify the column's head in the control's sort bar. By default, the ColumnAutoResize property is True.

property List.ColumnFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as Long

Retrieves the column 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.
Long	A long expression that indicates the column's index, or -1 if there is no column at the point. The property gets a negative value less or equal with 256, if the point is in the area between columns where the user can resize the column.

Use the ColumnFromPoint property to access the column from the point specified by the {X,Y} coordinates. The ColumnFromPoint property gets the index of the column when the cursor hovers the control's header bar. The X and Y coordinates are expressed in client coordinates, so a conversion must be done in case your coordinates are relative to the screen or to other window. **If the X parameter is -1 and Y parameter is -1 the ColumnFromPoint property determines the index of the column from the cursor.** Use the ItemFromPoint property to retrieve the item from cursor. The control fires the ColumnClick event when user clicks a column. Use the SortOnClick property to specify the operation that control odes when user clicks the control's header.

The following VB sample prints the caption of the column from the point:

```
Private Sub List1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With List1
    Dim c As Long
    c = .ColumnFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If (c >= 0) Then
      With .Columns(c)
        Debug.Print .Caption
      End With
    End If
  End With
End Sub
```

The following C++ sample prints the caption of the column from the point:

```
#include "Columns.h"
#include "Column.h"
void OnMouseMoveList1(short Button, short Shift, long X, long Y)
{
   long nColIndex = m_list.GetColumnFromPoint( X, Y );
   if ( nColIndex >= 0 )
   {
      CColumn column = m_list.GetColumns().GetItem( COleVariant( nColIndex ) );
      OutputDebugString( column.GetCaption() );
   }
}
```

The following VB.NET sample prints the caption of the column from the point:

```
Private Sub AxList1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseMoveEvent) Handles AxList1.MouseMoveEvent
   With AxList1
      Dim i As Integer = .get_ColumnFromPoint(e.x, e.y)
      If (i >= 0) Then
         With .Columns(i)
            Debug.WriteLine(.Caption)
         End With
      End If
   End With
End Sub
```

The following C# sample prints the caption of the column from the point:

```
private void axList1_MouseMoveEvent(object sender,
AxEXLISTLib._IListEvents_MouseMoveEvent e)
{
   int i = axList1.get_ColumnFromPoint( e.x,e.y );
   if ( i >= 0 )
      System.Diagnostics.Debug.WriteLine( axList1.Columns[i].Caption  );
}
```

The following VFP sample prints the caption of the column from the point:

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

with thisform.List1
   i = .ColumnFromPoint( x, y )
   if ( i >= 0 )
      wait window nowait .Columns(i).Caption
   endif
endwith
```

property List.Columns as Columns

Retrieves the control's column collection.

Type	Description
Columns	A Columns object that holds the control's columns collection.

Use the Columns property to access the Columns collection. Use the Columns collection to add, remove or change columns. Use the Add method to add a new column to the control. Use the Items property to access the control's items collection. Use the Add or PutItems method to add new items to the control. Use the DataSource property to add new columns and items to the control. Adding new items fails if the control has no columns.

property List.ColumnsAllowSizing as Boolean

Retrieves or sets a value that indicates whether a user can resize columns at run-time.

Type	Description
Boolean	A Boolean expression that indicates whether a user can resize columns at run-time.

By default, the ColumnsAllowSizing property is False. A column can be resized only if the AllowSizing property is True. Use the DrawGridLines property to show or hide the control's grid lines. Use the HeaderVisible property to show or hide the control's header bar. The HeaderAppearance property specifies the appearance of the column in the control's header bar

property List.ConditionalFormats as ConditionalFormats

Retrieves the conditional formatting collection.

Type	Description
ConditionalFormats	A ConditionalFormats object that indicates the control's ConditionalFormats collection.

The conditional formatting feature allows you to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. Use the Add method to format cells or items based on a formula. Use the Refresh method to refresh the control, if a change occurs in the conditional format collection. Use the Caption property indicates the cell's caption or value.

The conditional format feature may change the cells and items as follows:

- Bold property. Bolds the cell or items
- Italic property. Indicates whether the cells or items should appear in italic.
- StrikeOut property. Indicates whether the cells or items should appear in strikeout.
- Underline property. Underlines the cells or items
- Font property. Changes the font for cells or items.
- BackColor property. Changes the background color for cells or items, supports skins as well.
- ForeColor property. Changes the foreground color for cells or items.

The ApplyTo property specifies whether the ConditionalFormat object is applied to items or to a column.

property List.ContinueColumnScroll as Boolean

Retrieves or sets a value indicating whether the control will automatically scroll the visible columns by pixel or by column width.

Type	Description
Boolean	A boolean expression indicating whether the control will automatically scroll the visible columns by pixel or by column width.

By default, the columns are scrolled pixel by pixel. Use the ContinueColumnScroll to scroll horizontally the control column by column. Use the Visible property to hide a column. The ScrollBySingleLine property retrieves or sets a value that indicates whether the control scrolls the lines to the end, item by item. Use the ScrollBars property to hide the control's scroll bars. Use the ScrollPos property to scroll the control's content.

method List.Copy ()

Copies the control's content to the clipboard, in the EMF format.

Type	Description

Use the Copy method to copy the control's content to the clipboard, in Enhanced Metafile (EMF) format. The Enhanced Metafile format is a 32-bit format that can contain both vector information and bitmap information. This format is an improvement over the Windows Metafile Format and contains extended features, such as the following:

Built-in scaling information
Built-in descriptions that are saved with the file
Improvements in color palettes and device independence

The EMF format is an extensible format, which means that a programmer can modify the original specification to add functionality or to meet specific needs. You can paste this format to Microsoft Word, Excel, Front Page, Microsoft Image Composer and any application that know to handle EMF formats.

The Copy method copies the control's header if it's visible, and all visible items. The items are not expanded, they are listed in the order as they are displayed on the screen. Use the HeaderVisible property to show or hide the control's header. The background of the copied control is transparent.

The following VB sample saves the control's content to a EMF file, when user presses the CTRL+C key:

```
Private Sub List1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = vbKeyC) And Shift = 2 Then
      Clipboard.Clear
      List1.Copy
      SavePicture Clipboard.GetData(), App.Path & "\test.emf"
   End If
End Sub
```

Now, you can open your MS Windows Word application, and you can insert the file using the Insert\Picture\From File menu, or by pressing the CTRL+V key to paste the clipboard.

The following C++ function saves the clipboard's data (EMF format) to a picture file:

```
BOOL saveEMFtoFile( LPCTSTR szFileName )
{
```

```
BOOL bResult = FALSE;
if ( ::OpenClipboard( NULL ) )
{
    CComPtr<IPicture> spPicture;
    PICTDESC pictDesc = {0};
    pictDesc.cbSizeofstruct = sizeof(pictDesc);
    pictDesc.emf.hemf = (HENHMETAFILE)GetClipboardData( CF_ENHMETAFILE );
    pictDesc.picType = PICTYPE_ENHMETAFILE;
    if ( SUCCEEDED( OleCreatePictureIndirect( &pictDesc, IID_IPicture, FALSE,
(LPVOID*)&spPicture ) ) )
    {
        HGLOBAL hGlobal = NULL;
        CComPtr<IStream> spStream;
        if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal = GlobalAlloc( GPTR, 0 ), TRUE,
&spStream ) ) )
        {
            long dwSize = NULL;
            if ( SUCCEEDED( spPicture->SaveAsFile( spStream, TRUE, &dwSize ) ) )
            {
                USES_CONVERSION;
                HANDLE hFile = CreateFile( szFileName, GENERIC_WRITE, NULL, NULL,
CREATE_ALWAYS, NULL, NULL );
                if ( hFile != INVALID_HANDLE_VALUE )
                {
                    LARGE_INTEGER l = {NULL};
                    spStream->Seek(l, STREAM_SEEK_SET, NULL);
                    long dwWritten = NULL;
                    while ( dwWritten < dwSize )
                    {
                        unsigned long dwRead = NULL;
                        BYTE b[10240] = {0};
                        spStream->Read( &b, 10240, &dwRead );
                        DWORD dwBWritten = NULL;
                        WriteFile( hFile, b, dwRead, &dwBWritten, NULL );
                        dwWritten += dwBWritten;
                    }
                    CloseHandle( hFile );
```

```
                bResult = TRUE;
            }
        }
    }
  }
  CloseClipboard();
}
return bResult;
}
```

The following VB.NET sample copies the control's content to the clipboard (open the mspaint application and paste the clipboard, after running the following code):

```
Clipboard.Clear()
With AxList1
  .Copy()
End With
```

The following C# sample copies the control's content to a file (open the mspaint application and paste the clipboard, after running the following code):

```
Clipboard.Clear;
axList1.Copy();
```

property List.CopyTo (File as String) as Variant

Exports the control's view to an EMF file.

Type	Description
File as String	A String expression that indicates the name of the file being saved (EMF file). If present, the CopyTo property retrieves True, if the operation succeeded, else False it is failed. If the File parameter is missing or empty, the CopyTo property retrieves an one dimension safe array of bytes that contains the EMF content.
Variant	A boolean expression that indicates whether the File was successful saved, or a one dimension safe array of bytes, if the File parameter is empty string.

The CopyTo method copies the control's view to EMF files. Use the Copy method to copy the control's content to the clipboard. The Enhanced Metafile format is a 32-bit format that can contain both vector information and bitmap information. You can use the Export method to export the control's DATA in CSV format.

This format is an improvement over the Windows Metafile Format and contains extended features, such as the following

Built-in scaling information
Built-in descriptions that are saved with the file
Improvements in color palettes and device independence

The EMF format is an extensible format, which means that a programmer can modify the original specification to add functionality or to meet specific needs. You can paste this format to Microsoft Word, Excel, Front Page, Microsoft Image Composer and any application that know to handle EMF formats.

The following VB sample saves the control's content to a file:

```
If (List1.CopyTo("c:\temp\test.emf")) Then
   MsgBox "test.emf file created, open it using the mspaint editor."
End If
```

The following VB sample prints the EMF content (as bytes, File parameter is empty string):

```
Dim i As Variant
For Each i In List1.CopyTo("")
```

```
    Debug.Print i
Next
```

property List.CountLockedColumns as Long

Retrieves or sets a value indicating the number of locked columns. A locked column is not scrollable.

Type	Description
Long	A long expression that indicates the count of locked columns. The locked columns are fixed to left side.

The ExList control can group the control columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. Use BackColorLock property to change the background color of the control's locked area. Use ForeColorLock property to change the foreground color of the control's locked area. Use the Def(exCellBackColor) property to specify the background color for all cells in the column.

property List.DataSource as Object

Retrieves or sets a value that indicates the data source for object.

Type	Description
Object	An Object that defines the control's data. Currently, the control accepts ADO.Recordset, ADODB.Recordset objects, DAO recordsets

The **/COM** version provides ADO, ADODB and DAO database support. The DataSource property takes a recordset and add a column for each field found, and add a new item for each record in the recordset. Use the Visible property to hide a column. Use the Caption property to retrieves the value of the cell. Use the PutItems to load an array to the control. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. Use the DefaultItemHeight property before setting a DataSource property to specify the

The **/NET** version provides the following methods for data binding:

- *DataSource*, gets or sets the data source that the control is displaying data for. By default, this property is empty object. The DataSource property can be: DataTable, DataView, DataSet, DataViewManager, any component that implements the IListSource interface, or any component that implements the IList interface.
- *DataMember*, indicates a sub-list of the DataSource to show in the control. By default, this property is "". For instance, if DataSource property is a DataSet, the DataMember should indicates the name of the table to be loaded.

Click here to watch a movie on how to assign a data source to the control, in design mode, for /NET assembly.

The DataSource property can load all data in the memory or just visible records (virtual mode). The VirtualMode property indicates whether the control loads all records in memory or just visible records. If the VirtualMode property is False (by default), all records are loaded in memory. The user must call the VirtualMode property before setting the DataSource, else an error occurs. If the VirtualMode property is True, before specifying the DataSource, the control loads virtually the records, just visible records are loaded. Use the control's virtual mode when you require to display and edit large databases, and you don't want to load the entire database in memory. Aldo, running the virtual mode disables some features including sorting and filtering, like explained in the VirtualMode property. The control builds an internal object that implements the IUnboundHandler interface that provides data for the control, when running in virtual mode, so the UnboundHandler property is not empty.

The following template script loads virtually the Order table, using the Template feature of

the control (copy the following template and paste it to the control's WYSWYG Template editor) (the sample uses Jet.OLEDB provider to handle MDB files) :

```
Dim rs
VirtualMode = True
ColumnAutoResize = False
rs = CreateObject("ADOR.Recordset")
{
   Open("Orders","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExList\Sample\SAMPLE.MDB", 3, 3 )
}
DataSource = rs
ConditionalFormats
{
   Add("%1 > 4").Bold = True
   Add("%1 = 1 or %1 = 3")
   {
      Underline = True
      ForeColor = RGB(255,0,0)
      ApplyTo = 1
   }
}
```

or (the sample uses VFPOLEDB provider to handle DBF files)

```
Dim rs
VirtualMode = True
ColumnAutoResize = False
rs = CreateObject("ADODB.Recordset").Open("Select * from
Students","Provider=vfpoledb;Data Source=D:\Program Files\Microsoft Visual
Studio\Vfp98\Wizards\Template\Students And Classes\Data\STUDENTS AND
CLASSES.DBC;Collating Sequence=machine"1,1 )
DataSource = rs
```

Use the Caption property to retrieves the value of the cell. Use the PutItems to load an array to the control. If the VirtualMode property is False, the DetectAddNew detects when a new record is added to the recordset and updates the control's list so the new record is included. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a

formula.

The following VB sample binds an ADO recordset to the ExList control:

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Orders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Opens the table using static mode

Set List1.DataSource = rs
```

The DataSource clears the columns collection, and fill the record set into the list.

The following C++ sample binds a table to the control:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"

#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;

_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
{
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_list.BeginUpdate();
         m_list.SetColumnAutoResize( FALSE );
         m_list.SetDataSource( spRecordset );
```

```
        m_list.EndUpdate();
      }
    }
    catch ( _com_error& e )
    {
      AfxMessageBox( e.Description() );
    }
}
```

The #import statement imports definitions for ADODB type library, that's used to fill the control

property List.DefaultItemHeight as Long

Retrieves or sets a value that indicates the default item height.

Type	Description
Long	A long expression that indicates the default item's height.

The DefaultItemHeight property specifies the height of the items. Changing the property fails if the control contains already items. You can change the DefaultItemHeight property at design time, or at runtime, before adding any new items to the Items collection. Use the ItemHeight property to specify the height of a specified item. Use the ScrollBySingleLine property when using the items with different heights. Use the CellSingleLine property to specify whether the cell displays the caption using multiple lines.

property List.Description(Type as DescriptionTypeEnum) as String

Changes descriptions for control objects.

Type	Description
Type as DescriptionTypeEnum	A long expression that defines the part being changed.
String	A string value that indicates the part's description.

Use the Description property to customize the captions for control filter bar window. For instance, the Description(exFilterAll) = "(Include All)" changes the "(All)" item description in the filter bar window. Use the Description property to change the predefined strings in the filter bar window. Use FilterBarDropDownWidth property to specify the width of the drop down filter window.

property List.DetectAddNew as Boolean

Specifies whether the control detects when a new record is added to the bounded recordset.

Type	Description
Boolean	A boolean expression that indicates whether the control detects when a new record is added to the bounded recordset.

The DetectAddNew property detects adding new records to a recordset. Use the DataSource property to bound the control to a table. If the DetectAddNew property is True, and user adds a new record to the bounded recordset, the control automatically adds a new item to the control. The DetectAddNew property has effect only if the control is bounded to an ADO, ADODB recordset, using the DataSource property.

property List.DetectDelete as Boolean

Specifies whether the control detects when a record is deleted from the bounded recordset.

Type	Description
Boolean	A boolean expression that indicates whether the control detects when a record is deleted from the bounded recordset.

The property has effect only if the DataSource property points to a ADO recordset. If the DetectDelete property is True, the control is notified when a record is deleted, and updates the control.

property List.DrawGridLines as GridLinesEnum

Retrieves or sets a value that indicates whether the grid lines are visible or hidden.

Type	Description
GridLinesEnum	A GridLinesEnum expression that indicates whether the grid lines are visible or hidden.

Use the DrawGridLines property to add grid lines to the current view. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible. Use the HeaderVisible property to show or hide the control's header bar.

property List.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 ForeColor property to change the control's foreground color. Use the BackColor property to change the control's background color. Use the EnableItem to disable an item. Use the CellEnabled property to disable a cell. Use the Enabled property to disable a column. Use the SelectableItem property to specify whether an user can select an item.

method List.EndUpdate ()

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

Type	Description

The BeginUpdate and EndUpdate methods increases the speed of loading your items, by preventing painting the control when it suffers any change. Once that BeginUpdate method was called, you have to make sure that EndUpdate method will be called too.

The following VB sample prevents painting the control while the control loads data from a recordset:

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Orders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Opens the table using static mode

With List1
   .BeginUpdate
   For Each f In rs.Fields
      .Columns.Add f.Name
   Next
   .PutItems rs.GetRows()
   .EndUpdate
End With
```

The following C++ sample prevents refreshing the control while adding columns and items from an ADODB recordset:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"

#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;

_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
{
```

```
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_list.BeginUpdate();
         m_list.SetColumnAutoResize( FALSE );
         CColumns columns = m_list.GetColumns();
         long nCount = spRecordset->Fields->Count;
         if ( nCount > 0 )
         {
            // Adds the columns
            for ( long i = 0 ; i < nCount; i++ )
               columns.Add( spRecordset->Fields->Item[ i ]->Name );
            m_list.PutItems( &spRecordset->GetRows(-1), vtMissing );
         }
         m_list.EndUpdate();
      }
   }
   catch ( _com_error& e )
   {
      AfxMessageBox( e.Description() );
   }
}
```

The sample adds a column for each field in the recordset, and add a new items for each record. You can use the DataSource property to bind a recordset to the control. The #import statement imports definitions for ADODB type library, that's used to fill the control.

The following VB.NET sample prevents refreshing the control while adding columns and items:

```
With AxList1
   .BeginUpdate()
   With .Columns
      .Add("Column 1")
      .Add("Column 2")
   End With
   With .Items
      Dim iNewItem As Integer
      iNewItem = .Add("Item 1")
      .Caption(iNewItem, 1) = "SubItem 1"
      iNewItem = .Add("Item 2")
      .Caption(iNewItem, 1) = "SubItem 2"
   End With
   .EndUpdate()
End With
```

The following C# sample prevents refreshing the control while adding columns and items:

```
axList1.BeginUpdate();
EXLISTLib.Columns columns = axList1.Columns;
columns.Add("Column 1");
columns.Add("Column 2");
EXLISTLib.Items items = axList1.Items;
int iNewItem = items.Add("Item 1");
items.set_Caption(iNewItem, 1, "SubItem 1");
iNewItem = items.Add("Item 2");
items.set_Caption(iNewItem, 1, "SubItem 2");
axList1.EndUpdate();
```

The following VFP sample prevents refreshing the control while adding new columns and items:

```
thisform.List1.BeginUpdate()
with thisform.List1.Columns
   .Add("Column 1")
   .Add("Column 2")
endwith
```

```
with thisform.List1.Items
   local i
   i = .Add("Item 1")
   .Caption(i, 1) = "SubItem 1"
   i = .Add("Item 2")
   .Caption(i, 1) = "SubItem 2"
endwith
thisform.List1.EndUpdate()
```

property List.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 List.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 defines the result of the call.

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 background color of the control:

```
Debug.Print List1.ExecuteTemplate("BackColor")
```

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

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

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

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

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

method List.Export ([Destination as Variant], [Options as Variant])

Exports the control's data to a CSV or HTML format.

Type	Description
Destination as Variant	A String expression that specifies the file/format to be created. The Destination parameter indicates the format to be created as follows: • if "htm" or "html", the control returns the HTML format (including CSS style) • Any file-name that ends on ".htm" or ".html" creates the file with the HTML format inside • missing, empty, or any other case the Export exports the control's data in CSV format. No error occurs, if the Export method can not create the file.
Options as Variant	A String expression that specifies the options to be used when exporting the control's data, as explained bellow.

Return	Description
Variant	A String expression that indicates the format being exported. It could be CSV or HTML format based on the Destination parameter.

The Export method can export the control's DATA to a CSV or HTML format. The Export method can export a collection of columns from selected, visible, check or all items. By default, the control export all items, unless there is no filter applied on the control, where only visible items are exported. No visual appearance is saved in CSV format, instead the HTML format includes the visual appearance in CSS style.

The Options parameter consists a list of fields separated by | character, in the following order:

1. The first field could be **all**, **vis**, **sel** or **chk**, to export all, just visible, selected or checked items. The all option is used, if the field is missing. The **all** option displays all items, including the hidden or collapsed items. The **vis** option includes the visible items only, not including the child items of a collapsed item, or not-visible items (item's height is 0). The **sel** options lists the items being selected. The **chk** option lists all check and visible items. If chk option is used, the first column in the columns list should indicate the index of the column being queried for a check box state.
2. the second field indicates the column to be exported. All visible columns are exported,

if missing. The list of columns is separated by , character, and indicates the index of the column to be shown on the exported data. The first column in the list indicates the column being queried, if the option **chk** is used.

3. the third field indicates the character to separate the fields inside each exported line [tab character-if missing]. This field is valid, only when exporting to a CSV format
4. the forth field could be **ansi** or **unicode**, which indicates the character-set to save the control's content to Destination. For instance, Export(Destination,"|||unicode") saves the control's content to destination in UNICODE format (two-bytes per character). By default, the Export method creates an ANSI file (one-byte character)

The Destination parameter indicates the file to be created where exported date should be saved. For instance, Export("c:\temp\export.html") exports the control's DATA to export.html file in HTML format, or Export("","sel|0,1|;") returns the cells from columns 0, 1 from the selected items, to a CSV format using the ; character as a field separator.

The "CSV" refers to any file that:

- CSV stands for Comma Separated Value
- is plain text using a character set such as ASCII, Unicode,
- consists of records (typically one record per line),
- with the records divided into fields separated by delimiters (typically a single reserved character such as tab, comma, or semicolon; sometimes the delimiter may include optional spaces),
- where every record has the same sequence of fields

The "HTML" refers to any file that:

- HTML stands for HyperText Markup Language.
- is plain text using a character set such as ASCII, Unicode
- It's the way web pages are encoded to handle things like bold, italics and even color text red.

You can use the Copy/CopyTo to export the control's view to clipboard/EMF/BMP/JPG/PNG/GIF or PDF format.

property List.FilterBarBackColor as Color

Specifies the background color of the control's filter bar.

Type	Description
Color	A color expression that defines the background color for description of the control's filter. 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.

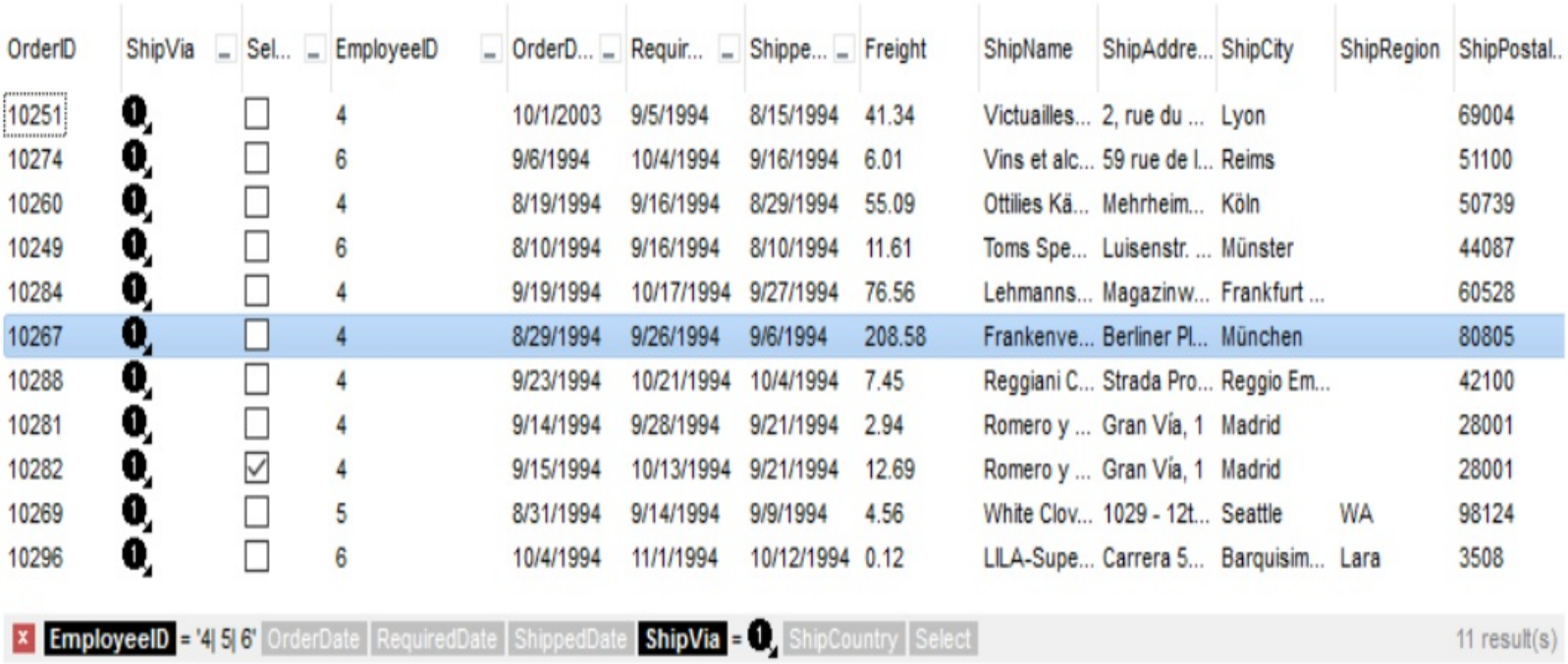
Use the FilterBarForeColor and FilterBarBackColor properties to define the colors used to paint the description for control's filter. Use the FilterBarHeight property to hide the control's filter bar header. Use the BackColor property to specify the control's background color. Use the BackColorLevelHeader property to specify the background color of the header when it displays multiple levels. Use the BackColorSortBar property to specify the background color of the control's sort bar.

property List.FilterBarCaption as String

Specifies the filter bar's caption.

Type	Description
String	A string value that defines the expression to display the control's filter bar.

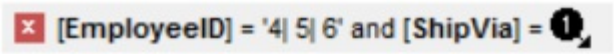
By default, the FilterBarCaption property is empty. You can use the FilterBarCaption property to define the way the filter bar's caption is being displayed. The FilterBarCaption is displayed on the bottom side of the control where the control's filter bar is shown. While the FilterBarCaption property is empty, the control automatically builds the caption to be displayed on the filter bar from all columns that participates in the filter using its name and values. For instance, if the control filters items based on the columns "EmployeeID" and "ShipVia", the control's filter bar caption would appear such as "[EmployeeID] = '...' and [ShipVia] = '...'". The FilterBarCaption property supports expressions as explained bellow.



For instance:

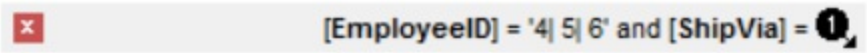
- "no filter", shows no filter caption all the time

- "" displays no filter bar, if no filter is applied, else it displays the current filter



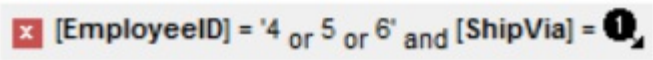
- "`<r>` + value", displays the current filter caption aligned to the right. You can include

the exFilterBarShowCloseOnRight flag into the FilterBarPromptVisible property to display the close button aligned to the right

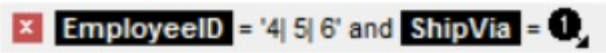


- "value replace ` and ` with `<fgcolor=FF0000> and </fgcolor>`", replace the AND keyword with a different foreground color

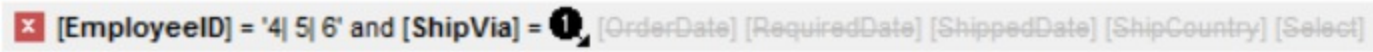
- "value replace ` and ` with `<off 4> and </off>` replace `|` with ` <off 4>or</off> ` replace ` ` with ` `", replaces the AND and | values



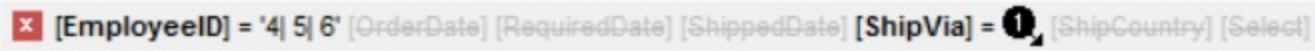
- "value replace `[` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `]` with ` </b></bgcolor></fgcolor>`", highlights the columns being filtered with a different background/foreground colors.



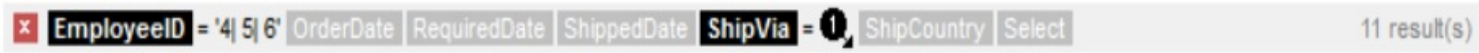
- "value + ` ` + available", displays the current filter, including all available columns to be filtered



- "allui" displays all available columns



- "((allui + `<fgcolor=808080>` + (matchitemcount < 0 ? ((len(allui) ? `` : ``) + `<r>` + abs(matchitemcount + 1) + ` result(s)`) : (`<r><fgcolor=808080>`+ itemcount + ` item(s)`))) replace `[<b>` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `</b>]` with ` </b></bgcolor></fgcolor>` replace `[<s>` with `<bgcolor=C0C0C0> <fgcolor=FFFFFF> ` replace `</s>]` with ` </bgcolor></fgcolor>`)" displays all available columns to be filtered with different background/foreground colors including the number of items/results



Use the FilterBarForeColor and FilterBarBackColor properties to define the colors used to paint the description for control's filter. Use the FilterBarHeight property to specify the

height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar. Use the Description property to define predefined strings in the filter bar caption. The VisibleItemCount property specifies the number of visible items in the list. The MatchItemCount property returns the number of matching items. The FilterBarPromptVisible property specifies whether how/where the control's filter/prompt is shown.

The FilterBarCaption method supports the following keywords, constants, operators and functions:

- **value** or **current** keyword returns the current filter as a string. At runtime the value may return a string such as "[EmployeeID] = '4| 5| 6' and [ShipVia] = <img>1</img>", so the control automatically applies HTML format, which you can change it. For instance, "upper(value)" displays the caption in uppercase or "value replace `<b>` with `<fgcolor=808080>` replace `</b>` with `</fgcolor>`" displays the column's name with a different foreground color.
- **itemcount** keyword returns the total number of items as indicated by Count property. At runtime the itemcount is a positive integer that indicates the count of all items. For instance, "value + `<r><fgcolor=808080>Total: ` + itemcount" includes in the filter bar the number of items aligned to the right.
- **visibleitemcount** keyword returns the number of visible items as indicated by VisibleItemCount property. At runtime, the visibleitemcount is a positive integer if no filter is applied, and negative if a filter is applied. If positive, it indicates the number of visible items. The visible items does not include child items of a collapsed item. If negative, a filter is applied, and the absolute value minus one, indicates the number of visible items after filter is applied. 0 indicates no visible items, while -1 indicates that a filter is applied, but no item matches the filter criteria. For instance, "value + `<r><fgcolor=808080>` + (visibleitemcount < 0 ? (`Result: ` + (abs(visibleitemcount) - 1)) : (`Visible: ` + visibleitemcount))" includes "Visible: " plus number of visible items, if no filter is applied or "Result: " plus number of visible items, if filter is applied, aligned to the right
- **matchitemcount** keyword returns the number of items that match the filter as indicated by MatchItemCount property. At runtime, the matchitemcount is a positive integer if no filter is applied, and negative if a filter is applied. If positive, it indicates the number of items within the control (Count property). If negative, a filter is applied, and the absolute value minus one, indicates the number of matching items after filter is applied. 0 indicates no visible items, while -1 indicates that a filter is applied, but no item matches the filter criteria. For instance, "value + `<r><fgcolor=808080>` + (matchitemcount < 0 ? (`Result: ` + (abs(matchitemcount) - 1)) : (`Visible: ` + matchitemcount))" includes "Visible: " plus number of visible items, if no filter is applied or "Result: " plus number of macthing items, if filter is applied, aligned to the right
- **leafitemcount** keyword returns the number of leaf items. A leaf item is an item with no child items. At runtime, the leafitemcount is a positive number that computes the

number of leaf items (expanded or collapsed). For instance, the "value + `<r> <fgcolor=808080><font ;6>` + leafitemcount" displays the number of leaf items aligned to the right with a different font and foreground color.

- **promptpattern** returns the pattern in the filter bar's prompt, as a string. The FilterBarPromptPattern specifies the pattern for the filter prompt. The control's filter bar prompt is visible, if the exFilterBarPromptVisible flag is included in the FilterBarPromptVisible property.
- **available** keyword returns the list of columns that are not currently part of the control's filter, but are available to be filtered. A column is available to be filtered, if the DisplayFilterButton property of the Column object, is True. At runtime, the available keyword may return a string such as "<fgcolor=C0C0C0>[<s>OrderDate</s>] <fgcolor> </fgcolor>[<s>RequiredDate</s>]<fgcolor> </fgcolor> [<s>ShippedDate</s>]<fgcolor> </fgcolor>[<s>ShipCountry</s>]<fgcolor> </fgcolor> [<s>Select</s>]</fgcolor>", so the control automatically applies HTML format, which you can change it. For instance, "value + ` ` + available", displays the current filter, including all available columns to be filtered. For instance, the "value + `<r>` + available replace `C0C0C0` with `FF0000`" displays the available columns aligned to the right with a different foreground color.
- **allui** keyword returns the list of columns that are part of the current filter and available columns to be filtered. A column is available to be filtered, if the DisplayFilterButton property of the Column object, is True. At runtime, the allui keyword may return a string such as "[EmployeeID] = '4| 5| 6'<fgcolor> </fgcolor><fgcolor=C0C0C0> [<s>OrderDate</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0> [<s>RequiredDate</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0> [<s>ShippedDate</s>]</fgcolor><fgcolor> </fgcolor>[ShipVia] = <img>1</img><fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>ShipCountry</s>]</fgcolor> <fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>Select</s>]</fgcolor>", so the control automatically applies HTML format, which you can change it. For instance, "allui", displays the current filter, including all available columns to be filtered. For instance, the "((allui + `<fgcolor=808080>` + (matchitemcount < 0 ? ((len(allui) ? `` : ``) + `<r>` + abs(matchitemcount + 1) + ` result(s)`) : (`<r><fgcolor=808080>`+ itemcount + ` item(s)`))) replace `[<b>` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `</b>]` with ` </b></bgcolor></fgcolor>` replace `[<s>` with `<bgcolor=C0C0C0> <fgcolor=FFFFFF> ` replace `</s>]` with ` </bgcolor></fgcolor>`)" displays all available columns to be filtered with different background/foreground colors including the number of items/results
- **all** keyword returns the list of all columns (visible or hidden) no matter if the DisplayFilterButton property is True or False. At runtime, the all keyword may return a string such as "<fgcolor=C0C0C0>[<s>OrderID</s>]</fgcolor><fgcolor> </fgcolor> [EmployeeID] = '4| 5| 6'<fgcolor> </fgcolor><fgcolor=C0C0C0> [<s>OrderDate</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0> [<s>RequiredDate</s>]</fgcolor><fgcolor>", so the control automatically applies HTML format, which you can change it. For instance, "all", displays the current filter,

including all other columns. For instance, the "((all + `<fgcolor=808080>` + (matchitemcount < 0 ? ((len(allui) ? `` : ``) + `<r>` + abs(matchitemcount + 1) + ` result(s)`) : (`<r><fgcolor=808080>`+ itemcount + ` item(s)`))) replace `[<b>` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `</b>]` with ` </b></bgcolor> </fgcolor>` replace `[<s>` with `<bgcolor=C0C0C0><fgcolor=FFFFFF> ` replace `</s>]` with ` </bgcolor></fgcolor>`)" displays all columns with different background/foreground colors including the number of items/results

Also, the FilterBarCaption property supports predefined constants and operators/functions as described here.

Also, the FilterBarCaption 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.

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

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

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

lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

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

known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

property List.FilterBarDropDownHeight as Double

Specifies the height of the drop down filter window proportionally with the height of the control's list.

Type	Description
Double	A double expression that indicates the height of the drop down filter window.

Use the FilterBarDropDownHeight property to specify the height of the drop down window filter window. Use the DisplayFilterButton property to display a filter button to the column's caption. BY default, the FilterBarDropDownHeight property is 0.5. It means, the height of the drop down filter window is half of the height of the control's list. Use the Description property to define predefined strings in the filter bar. Use FilterBarDropDownWidth property to specify the width of the drop down filter window.

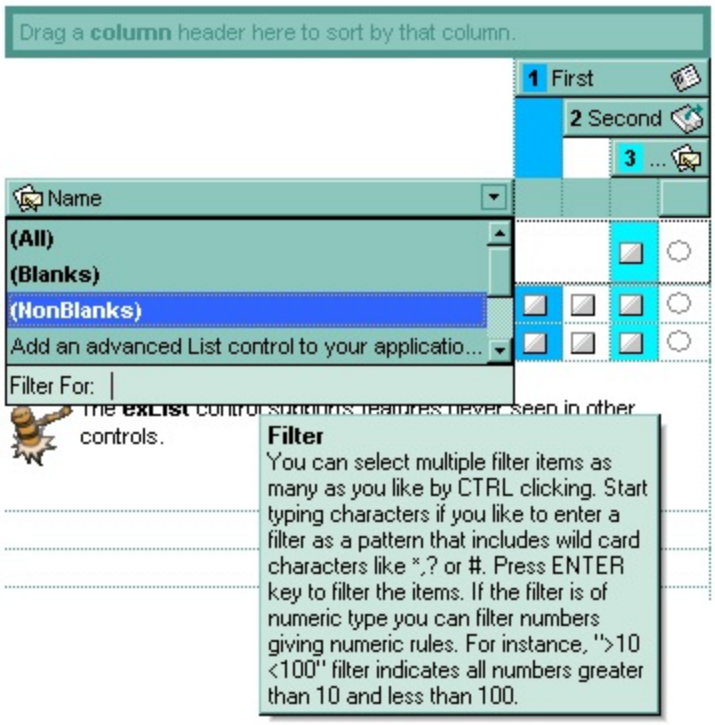
If the FilterBarDropDownHeight property is negative, the absolute value of the FilterBarDropDownHeight property indicates the height of the drop down filter window in pixels. In this case, the height of the drop down filter window is not proportionally with the height of the control's list area. For instance, the following sample specifies the height of the drop down filter window being 100 pixels:

```
With List1
   .FilterBarDropDownHeight = -100
End With
```

If the FilterBarDropDownHeight property is greater than 0, it indicates the height of the drop down filter window proportionally with the height of the control's height list. For instance, the following sample specifies the height of the drop down filter window being the same with the height of the control's list area:

```
With List1
   .FilterBarDropDownHeight = 1
End With
```

The drop down filter window always include an item.


Drag a column header here to sort by that column.
1 First
2 Second
3 ...
Name
(All)
(Blanks)
(NonBlanks)
Add an advanced List control to your applicatio...
Filter For:
controls.
Filter
You can select multiple filter items as many as you like by CTRL clicking. Start typing characters if you like to enter a filter as a pattern that includes wild card characters like *,? or #. Press ENTER key to filter the items. If the filter is of numeric type you can filter numbers giving numeric rules. For instance, ">10 <100" filter indicates all numbers greater than 10 and less than 100.

property List.FilterBarFont as IFontDisp

Retrieves or sets the font for control's filter bar.

Type	Description
IFontDisp	A font object that indicates the font used to paint the description for control's filter.

Use the FilterBarFont property to specify the font for the control's filter bar object. Use the Font property to set the control's font. Use the FilterBarHeight property to specify the height of the filter bar. Use the FilterBarCaption property to define the control's filter bar caption. Use the Refresh method to refresh the control.

The following VB sample assigns by code a new font to the filter bar control:

```
With List1
   With .FilterBarFont
      .Name = "Tahoma"
   End With
   .Refresh
End With
```

The following C++ sample assigns by code a new font to the filter bar control:

```
COleFont font = m_list.GetFilterBarFont();
font.SetName( "Tahoma" );
m_list.Refresh();
```

the C++ sample requires definition of COleFont class (#include "Font.h")

The following VB.NET sample assigns by code a new font to the filter bar control:

```
With AxList1
   Dim font As System.Drawing.Font = New System.Drawing.Font("Tahoma", 10,
FontStyle.Regular, GraphicsUnit.Point)
   .FilterBarFont = font
   .CtlRefresh()
End With
```

The following C# sample assigns by code a new font to the filter bar control:

```
System.Drawing.Font font = new System.Drawing.Font("Tahoma", 10, FontStyle.Regular);
axList1.FilterBarFont = font;
axList1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the filter bar control:

```
with thisform.List1.Object
   .FilterBarFont.Name = "Tahoma"
   .Refresh()
endwith
```

The following Template sample assigns by code a new font to the filter bar control:

```
FilterBarFont
{
   Name = "Tahoma"
}
```

property List.FilterBarForeColor as Color

Specifies the foreground color of the control's filter bar.

Type	Description
Color	A color expression that defines the foreground color of the description of the control's filter.

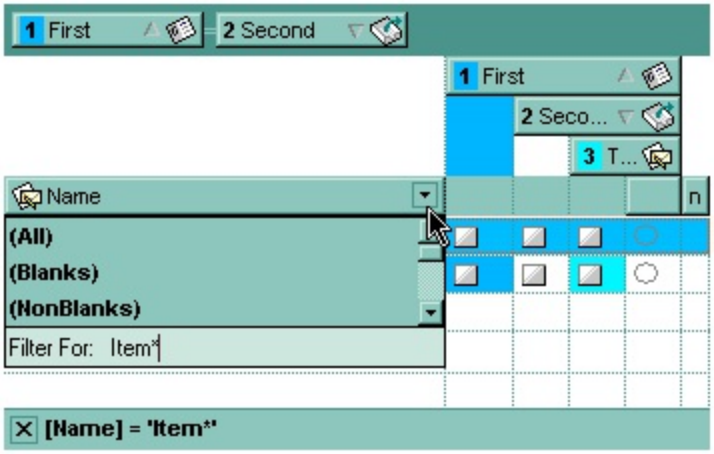
Use the FilterBarForeColor and FilterBarBackColor properties to define colors used to paint the description of the control's filter. Use the FilterBarFont property to specify the filter bar's font. Use the FilterBarCaption property to specify the caption of the control's filter bar.

property List.FilterBarHeight as Long

Specifies the height of the control's filter bar. If the value is less than 0, the filter bar is automatically resized to fit its description.

Type	Description
Long	A long expression that indicates the height of the filter bar.

The filter bar status defines the control's filter description. If the FilterBarHeight property is less than 0 the control automatically updates the height of the filter's description to fit in the control's client area. If the FilterBarHeight property is zero the filter's description is hidden. If the FilterBarHeight property is grater than zero it defines the height in pixels of the filter's description. Use the ClearFilter method to clear the control's filter. Use the FilterBarCaption property to define the control's filter bar caption. Use the FilterBarFont property to specify the font for the control's filter bar. Use the ShowFilter method to show programmatically the column's drop down filter window.



property List.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 FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered. Use the FilterBarCaption property to change the caption in the filter bar once a new filter is applied. The FilterBarFont property specifies the font to be used in the filter bar. The FilterBarBackColor property specifies the background color or the visual aspect of the control's filter bar. The FilterBarForeColor property specifies the foreground color or the control's filter bar.

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.

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

  - exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
  - e64, encodes in BASE64 the HTML text to be shown once the user clicks the

anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu " that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY string encodes the "<fgcolor 808080>show lines<a>-</fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part

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

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

gradient-center

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

outlined

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

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

shadow

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

outline anti-aliasing

The FilterBarPrompt property has effect only if:

- FilterBarPromptVisible property is True
- FilterBarPromptPattern property is Empty.

property List.FilterBarPromptColumns as Variant

Specifies the list of columns to be used when filtering using the prompt.

Type	Description
Variant	A long expression that indicates the index of the column to apply the filter prompt, a string expression that specifies the list of columns (indexes) separated by comma to apply the filter prompt, or a safe array of long expression that specifies the indexes of the columns to apply the filter. 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.

By default, the FilterBarPromptColumns property is -1. If the FilterBarPromptColumns property is -1, the filter prompt is applied for all columns, visible or hidden. Use the FilterBarPromptColumns property to specify the list of columns to apply the filter prompt pattern. The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way. The FilterBarPromptType property specifies the type of filtering when the user edits the prompt in the filter bar.

property List.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. The FilterBarPromptColumns specifies the list of columns to be used when filtering. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way.

property List.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. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered. Use the FilterBarCaption property to change the caption in the filter bar once a new filter is applied.

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

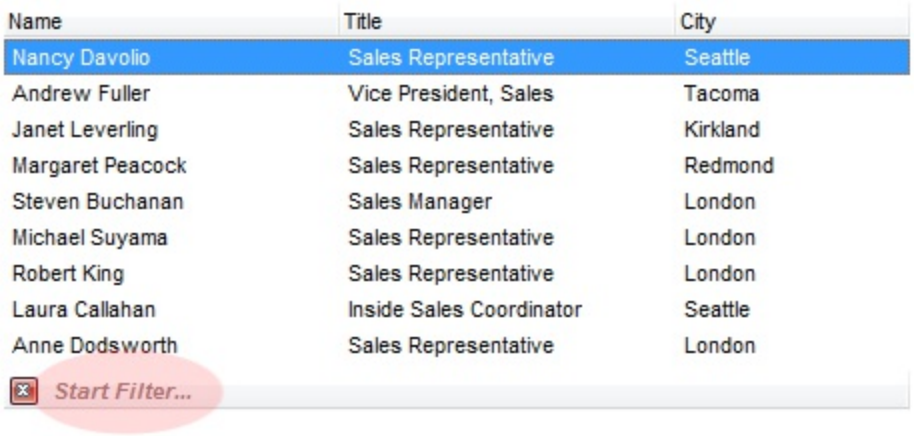
property List.FilterBarPromptVisible as FilterBarVisibleEnum

Shows or hides the control's filter bar including filter prompt.

Type	Description
FilterBarVisibleEnum	A FilterBarVisibleEnum expression that defines the way the control's filter bar is shown.

By default, The FilterBarPromptVisible property is exFilterBarHidden. 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. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way. The FilterBarCaption property defines the caption to be displayed on the control's filter bar. The FilterBarPromptType property specifies the type of filtering when the user edits the prompt in the filter bar. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered.

The following screen show shows the filter prompt:



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

Name	Title	City
Steven Buchanan	Sales Manager	London
Michael Suyama	Sales Representative	London
Robert King	Sales Representative	London
Anne Dodsworth	Sales Representative	London

london

property List.FilterCriteria as String

Retrieves or sets the filter criteria.

Type	Description
String	A string expression that indicates the filter criteria.

By default, the FilterCriteria property is empty. Use the FilterCriteria property to specify whether you need to filter items using OR, NOT operators between columns. If the FilterCriteria property is empty, or not valid, the filter uses the AND operator between columns. Use the FilterCriteria property to specify how the items are filtered.

The FilterCriteria property supports the following operators:

- **not** operator (unary operator)
- **and** operator (binary operator)
- **or** operator (binary operator)

Use the **(** and **)** parenthesis to define the order execution in the clause, if case. The operators are listed in their priority order. The **%** character precedes the index of the column (zero based), and indicates the column. For instance, **%0 or %1** means that OR operator is used when both columns are used, and that means that you can filter for values that are in a column or for values that are in the second columns. If a column is not listed in the FilterCriteria property, and the user filters values by that column, the AND operator is used by default. For instance, let's say that we have three columns, and FilterCriteria property is "%0 or %1". If the user filter for all columns, the filter clause is equivalent with (%0 or %1) and %2, and it means all that match the third column, and is in the first or the second column.

Use the Filter and FilterType properties to define a filter for a column. The ApplyFilter method should be called to update the control's content after changing the Filter or FilterType property, in code! Use the DisplayFilterButton property to display a drop down button to filter by a column.

property List.Font as IFontDisp

Retrieves or sets the control's font.

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

Use the Font property to change the control's font . Use the FilterBarFont property to assign a different font for the control's filter bar. Use the Refresh method to refresh the control. Use the BeginUpdate and EndUpdate method to maintain performance while adding new columns or items. Use the DefaultItemHeight property to specify the default height for all items in the control.

The following VB sample assigns by code a new font to the control:

```
With List1
   With .Font
      .Name = "Tahoma"
   End With
   .Refresh
End With
```

The following C++ sample assigns by code a new font to the control:

```
COleFont font = m_list.GetFont();
font.SetName( "Tahoma" );
m_list.Refresh();
```

the C++ sample requires definition of COleFont class (#include "Font.h")

The following VB.NET sample assigns by code a new font to the control:

```
With AxList1
   Dim font As System.Drawing.Font = New System.Drawing.Font("Tahoma", 10,
FontStyle.Regular, GraphicsUnit.Point)
   .Font = font
   .CtlRefresh()
End With
```

The following C# sample assigns by code a new font to the control:

```
System.Drawing.Font font = new System.Drawing.Font("Tahoma", 10, FontStyle.Regular);
axList1.Font = font;
axList1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the control:

```
with thisform.List1.Object
   .Font.Name = "Tahoma"
   .Refresh()
endwith
```

The following Template sample assigns by code a new font to the control:

```
Font
{
   Name = "Tahoma"
}
```

property List.ForeColor as Color

Retrieves or sets a value that indicates the control's foreground color.

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

The ForeColor property changes the foreground color of the control's scrolled area. The ExList control can group the columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. To change the background color of the control's locked area use BackColorLock property. Use the CellForeColor property to specify the cell's foreground color. Use the ItemForeColor property to specify the item's foreground color.

property List.ForeColorHeader as Color

Specifies the header's foreground color.

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

Use the BackColorHeader and ForeColorHeader properties to customize the control's header. Use the Font property to change the control's font. Use the Add method to add new columns to the control. Use the HeaderVisible property to hide the control's header bar.

property List.ForeColorLock as Color

Retrieves or sets a value that indicates the control's foreground color for the locked area.

Type	Description
Color	A color expression that indicates the foreground color of the control's un-scrolled area

The ExList control can group the control columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. Use BackColorLock property to change the background color of the control's locked area . Use the Def(exCellForeColor) property to specify the foreground color for all cells in the column.

property List.ForeColorSortBar as Color

Retrieves or sets a value that indicates the sort bar's foreground color.

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

Use the ForeColorSortBar property to specify the foreground color of the caption in the control's sort bar. Use the SortBarVisible property to show the control's sort bar. Use the SortBarCaption property to specify the caption of the sort bar, when the control's sort bar contains no columns. Use the BackColorSortBar property to specify the background color of the control's sort bar. Use the BackColorSortBarCaption property to specify the caption's background color in the control's sort bar. Use the ForeColor property to specify the control's foreground color. Use the ForeColorHeader property to specify the background color of the control's header bar.

method List.FormatABC (Expression as String, [A as Variant], [B as Variant], [C as Variant])

Formats the A,B,C values based on the giving expression and returns the result.

Type	Description
Expression as String	A String that defines the expression to be evaluated.
A as Variant	A VARIANT expression that indicates the value of the A keyword.
B as Variant	A VARIANT expression that indicates the value of the B keyword.
C as Variant	A VARIANT expression that indicates the value of the C keyword.
Return	**Description**
Variant	A VARIANT expression that indicates the result of the evaluation the List.

The FormatABC method formats the A,B,C values based on the giving expression and returns the result.

For instance:

- "A + B + C", adds / concatenates the values of the A, B and C
- "value MIN 0 MAX 99", limits the value between 0 and 99
- "value format ``", formats the value with two decimals, according to the control's panel setting
- "date(`now`)" returns the current time as double

The FormatABC method supports the following keywords, constants, operators and functions:

- **A** or **value** keyword, indicates a variable A whose value is giving by the A parameter
- **B** keyword, indicates a variable B whose value is giving by the B parameter
- **C** keyword, indicates a variable C whose value is giving by the C parameter

This property/method supports predefined constants and operators/functions as described here.

property List.FormatAnchor(New as Boolean) as String

Specifies the visual effect for anchor elements in HTML captions.

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

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

method List.FreezeEvents (Freeze as Boolean)

Prevents the control to fire any event.

Type	Description
Freeze as Boolean	A Boolean expression that specifies whether the control' events are froze or unfroze

The FreezeEvents(True) method freezes the control's events until the FreezeEvents(False) method is called. You can use the FreezeEvents method to improve performance of the control while loading data into it.

property List.FullRowSelect as Boolean

Enables full-row selection in the control.

Type	Description
Boolean	A boolean expression that indicates whether the full-row selection is enabled or disabled.

The FullRowSelect property specifies whether the selection spans the entire width of the control. The column pointed by the SelectColumnIndex specifies the column where the selected cell is marked. Use the SelectItem property to select programmatically an item. Use the SingleSel property to allow multiple items selection.

method List.GetItems (Options as Variant)

Gets the collection of items into a safe array,

Type	Description
Options as Variant	Specifies a long expression as follows: • If 1, the GetItems method returns the one-dimensional array of indexes of items in the control as they are displayed (sorted, filtered) • else the GetItems method gets a safe array that holds the items (values in the cells) in the control.

Return	Description
Variant	A safe array that holds the items in the control. The safe array being returned has two dimensions. The first dimension holds the collection of columns, and the second holds the cells.

The GetItems method to get a safe array that holds the items in the control. The GetItems method gets the items as they are displayed, sorted and filtered. Use the PutItems method to load an array to the control. The method returns nothing if the control has no columns or items. Use the Items property to access the items collection. You can use the GetItems(1) method to get the list of indexes for the items as they are displayed, sorted and filtered. The GetItems method returns an empty expression (VT_EMPTY), if there is no items in the result.

The following VB sample lists the indexes of the items in the control:

```
Dim v As Variant
For Each v In List1.GetItems(1)
   Debug.Print v
Next
```

The following VB sample gets the items from a control and put them to the second one:

```
With List2
   .BeginUpdate
      .Columns.Clear
      Dim c As EXLISTLibCtl.Column
      For Each c In List1.Columns
         .Columns.Add c.Caption
```

```
    Next
    .PutItems List1.GetItems
  .EndUpdate
End With
```

The following C++ sample gets the items from a control an put to the second one:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_list2.BeginUpdate();
  CColumns columns = m_list.GetColumns(), columns2 = m_list2.GetColumns();
  for ( long i = 0; i < columns.GetCount(); i++ )
    columns2.Add( columns.GetItem( COleVariant( i ) ).GetCaption() );
  COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
  COleVariant vtItems = m_list.GetItems( vtMissing );
  m_list2.PutItems( &vtItems, vtMissing );
m_list2.EndUpdate();
```

The following C# sample gets the items from a control and put them to a second one:

```
axList2.BeginUpdate();
for (int i = 0; i < axList1.Columns.Count; i++)
  axList2.Columns.Add(axList1.Columns[i].Caption);
object vtItems = axList1.GetItems("");
axList2.PutItems(ref vtItems);
axList2.EndUpdate();
```

The following VB.NET sample gets the items from a control and put them to a second one:

```
With AxList2
  .BeginUpdate()
  Dim j As Integer
  For j = 0 To AxList1.Columns.Count - 1
    .Columns.Add(AxList1.Columns(j).Caption)
  Next
  Dim vtItems As Object
  vtItems = AxList1.GetItems("")
  .PutItems(vtItems)
```

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

The following VFP sample gets the items from a control and put them to a second one:

```
local i
with thisform.List2
   .BeginUpdate()
   for i = 0 to thisform.List1.Columns.Count - 1
      .Columns.Add( thisform.List1.Columns(i).Caption )
   next
   local array vtItems[1]
   vtItems = thisform.List1.GetItems("")
   .PutItems( @vtItems )
   .EndUpdate()
endwith
```

property List.GridLineColor as Color

Specifies the grid line color.

Type	Description
Color	A color expression that indicates the color of the grid lines.

Use the GridLineColor property to specify the color for grid lines. Use the DrawGridLines property to show the grid lines. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible. Use the HeaderVisible property to show or hide the control's header bar.

property List.GridLineStyle as GridLinesStyleEnum

Specifies the style for gridlines in the list part of the control.

Type	Description
GridLinesStyleEnum	A GridLinesStyleEnum expression that specifies the style to show the control's horizontal or vertical lines.

By default, the GridLineStyle property is exGridLinesDot. The GridLineStyle property has effect only if the DrawGridLines property is not zero. The GridLineStyle property can be used to specify the style for horizontal or/and vertical grid lines.

The following VB sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = GridLinesStyleEnum.exGridLinesHDash Or
GridLinesStyleEnum.exGridLinesVSolid
```

The following VB/NET sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXLISTLib.GridLinesStyleEnum.exGridLinesHDash Or
exontrol.EXLISTLib.GridLinesStyleEnum.exGridLinesVSolid
```

The following C# sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXLISTLib.GridLinesStyleEnum.exGridLinesHDash |
exontrol.EXLISTLib.GridLinesStyleEnum.exGridLinesVSolid;
```

The following Delphi sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle := Integer(EXLISTLib.GridLinesStyleEnum.exGridLinesHDash) Or
Integer(EXLISTLib.GridLinesStyleEnum.exGridLinesVSolid);
```

The following VFP sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = 36
```

property List.HeaderAppearance as AppearanceEnum

Retrieves or sets a value that indicates the header's appearance.

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the header bar appearance.

Use the HeaderAppearance property to change the appearance of the control's header bar. Use the HeaderVisible property to hide the control's header bar. Use the Appearance property to specify the control's appearance. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible.

property List.HeaderHeight as Long

Retrieves or sets a value indicating control's header height.

Type	Description
Long	A long expression that indicates the height of the control's header bar.

Use the HeaderHeight property to change the height of the control's header bar. Use the HeaderVisible property to hide the control's header bar. Use the LevelKey property to display the control's header bar using multiple levels. If the control displays the header bar using multiple levels the HeaderHeight property gets the height in pixels of a single level in the header bar. The control's header displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the HTMLCaption property to display multiple lines in the column's caption. Use the Add method to add new columns to the control. The HeaderSingleLine property specifies whether the header displays captions using multiple lines.

For instance, the following VB sample displays the control's header bar using multiple lines:

```
With List1
    .BeginUpdate
      .HeaderHeight = 32
      With .Columns.Add("Line 1" & vbCrLf & "Line 2")
        .HeaderMultiLine = True
      End With
      With .Columns.Add("Line 1" & vbCrLf & "Line 2")
        .HeaderMultiLine = True
      End With
    .EndUpdate
End With
```

The following C++ sample displays a header bar using multiple lines:

```
#include "Columns.h"
#include "Column.h"
m_list.BeginUpdate();
m_list.SetHeaderHeight( 32 );
m_list.SetHeaderVisible( TRUE );
CColumn column1( V_DISPATCH( &m_list.GetColumns().Add( "Column 1" ) ) );
  column1.SetHTMLCaption( "Line1<br>Line2" );
```

```
CColumn column2( V_DISPATCH( &m_list.GetColumns().Add( "Column 2" ) ) );
   column2.SetHTMLCaption( "Line1<br>Line2" );
m_list.EndUpdate();
```

The following VB.NET sample displays a header bar using multiple lines:

```
With AxList1
   .BeginUpdate()
   .HeaderVisible = True
   .HeaderHeight = 32
   With .Columns.Add("Column 1")
      .HTMLCaption = "Line1<br>Line2"
   End With
   With .Columns.Add("Column 2")
      .HTMLCaption = "Line1<br>Line2"
   End With
   .EndUpdate()
End With
```

The following C# sample displays a header bar using multiple lines:

```
axList1.BeginUpdate();
axList1.HeaderVisible = true;
axList1.HeaderHeight = 32;
EXLISTLib.Column column1 = axList1.Columns.Add("Column 1") as EXLISTLib.Column ;
column1.HTMLCaption = "Line1<br>Line2";
EXLISTLib.Column column2 = axList1.Columns.Add("Column 2") as EXLISTLib.Column;
column2.HTMLCaption = "Line1<br>Line2";
axList1.EndUpdate();
```

The following VFP sample displays a header bar using multiple lines:

```
with thisform.List1
   .BeginUpdate()
   .HeaderVisible = .t.
   .HeaderHeight = 32
   with .Columns.Add("Column 1")
      .HTMLCaption = "Line1<br>Line2"
   endwith
```

```
    with .Columns.Add("Column 2")
        .HTMLCaption = "Line1<br>Line2"
    endwith
    .EndUpdate()
endwith
```

property List.HeaderSingleLine as Boolean

Specifies whether the control resizes the columns header and wraps the captions in single or multiple lines.

Type	Description
Boolean	A boolean expression that specifies whether the header displays single or multiple lines.

By defauly, the HeaderSingleLine property is True. If the HeaderSingleLine property is False the control breaks the column's caption as soon as the user resizes the column. **In this case the HeaderHeight property specifies the maximum height of the control's header.** The initial height is computed based on the control's Font property. The Caption property specifies the caption of the column being displayed in the control's header. The HTMLCaption property specifies the HTML caption of the column being displayed in the column's header. Use the LevelKey property to display the control's header on multiple levels.

property List.HeaderVisible as Boolean

Retrieves or sets a value that indicates whether the the list's header is visible or hidden.

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

By default, the HeaderVisible property is True. Use the HeaderVisible property to hide the control's header bar. Use the HeaderAppearance property to change the header bar's appearance. Use the BackColorHeader and ForeColorHeader properties to customize the control's header. Use the BackColorLevelHeader property to specify the background color of the header when it displays multiple levels. Use the HeaderHeight property to specify the height of the control's header bar. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible.

property List.HideSelection as Boolean

Returns a value that determines whether selected item appears highlighted when a control loses the focus.

Type	Description
Boolean	A boolean expression that determines whether selected item appears highlighted when a control loses the focus.

By default, the HideSelection property is False. You can use this property to indicate which item is highlighted while another form or a dialog box has the focus. Use the SelForeColor and SelBackColor property to customize the colors for the selected items in the control. Use the SelectItem property to programmatically select an item. Use the SelectedItem and SelectCount property to retrieve the list of selected items. Use the SelectableItem property to specify whether an items can be selected.

property List.HotBackColor as Color

Retrieves or sets a value that indicates the hot-tracking background color.

Type	Description
Color	A color expression that indicates the background color for item from the cursor (hovering the item). 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 HotBackColor property is 0, which means that the HotBackColor property has no effect. Use the HotBackColor property on a non-zero value to highlight the item from the cursor. The HotForeColor property specifies the foreground color to highlight the item from the cursor. The ItemFromPoint property gets the item from the cursor. The SelBackColor property specifies the selection background color. The SelBackMode property specifies the way the selected items are shown in the control.

The following sample displays a different background color mouse passes over an item.

VBA

```
With List1
   .BeginUpdate
   .Columns.Add "Def"
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .Add "Item A"
      .Add "Item B"
      .Add "Item C"
   End With
   .EndUpdate
End With
```

VB6

```
With List1
   .BeginUpdate
   .Columns.Add "Def"
```

```
    .HotBackColor = RGB(0,0,128)
    .HotForeColor = RGB(255,255,255)
    With .Items
        .Add "Item A"
        .Add "Item B"
        .Add "Item C"
    End With
    .EndUpdate
End With
```

VB.NET

```
With Exlist1
    .BeginUpdate()
    .Columns.Add("Def")
    .HotBackColor = Color.FromArgb(0,0,128)
    .HotForeColor = Color.FromArgb(255,255,255)
    With .Items
        .Add("Item A")
        .Add("Item B")
        .Add("Item C")
    End With
    .EndUpdate()
End With
```

VB.NET for /COM

```
With AxList1
    .BeginUpdate()
    .Columns.Add("Def")
    .HotBackColor = RGB(0,0,128)
    .HotForeColor = RGB(255,255,255)
    With .Items
        .Add("Item A")
        .Add("Item B")
        .Add("Item C")
    End With
    .EndUpdate()
```

```
End With
```

C++

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

   #import <ExList.dll>
   using namespace EXLISTLib;
*/
EXLISTLib::IListPtr spList1 = GetDlgItem(IDC_LIST1)->GetControlUnknown();
spList1->BeginUpdate();
spList1->GetColumns()->Add(L"Def");
spList1->PutHotBackColor(RGB(0,0,128));
spList1->PutHotForeColor(RGB(255,255,255));
EXLISTLib::IItemsPtr var_Items = spList1->GetItems();
   var_Items->Add("Item A");
   var_Items->Add("Item B");
   var_Items->Add("Item C");
spList1->EndUpdate();
```

C++ Builder

```
List1->BeginUpdate();
List1->Columns->Add(L"Def");
List1->HotBackColor = RGB(0,0,128);
List1->HotForeColor = RGB(255,255,255);
Exlistlib_tlb::IItemsPtr var_Items = List1->Items;
   var_Items->Add(TVariant("Item A"));
   var_Items->Add(TVariant("Item B"));
   var_Items->Add(TVariant("Item C"));
List1->EndUpdate();
```

C#

```
exlist1.BeginUpdate();
exlist1.Columns.Add("Def");
```

```
exlist1.HotBackColor = Color.FromArgb(0,0,128);
exlist1.HotForeColor = Color.FromArgb(255,255,255);
exontrol.EXLISTLib.Items var_Items = exlist1.Items;
   var_Items.Add("Item A");
   var_Items.Add("Item B");
   var_Items.Add("Item C");
exlist1.EndUpdate();
```

JavaScript

```
<OBJECT classid="clsid:CD481F4D-2D25-4759-803F-752C568F53B7" id="List1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   List1.BeginUpdate()

   List1.Columns.Add("Def")

   List1.HotBackColor = 8388608

   List1.HotForeColor = 16777215

   var var_Items = List1.Items

      var_Items.Add("Item A")

      var_Items.Add("Item B")

      var_Items.Add("Item C")

   List1.EndUpdate()

</SCRIPT>
```

C# for /COM

```
axList1.BeginUpdate();
axList1.Columns.Add("Def");
```

```
axList1.HotBackColor = Color.FromArgb(0,0,128);
axList1.HotForeColor = Color.FromArgb(255,255,255);
EXLISTLib.Items var_Items = axList1.Items;
   var_Items.Add("Item A");
   var_Items.Add("Item B");
   var_Items.Add("Item C");
axList1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   COM com_Items

   anytype var_Items




   super()


   exlist1.BeginUpdate()

   exlist1.Columns().Add("Def")

   exlist1.HotBackColor(WinApi::RGB2int(0,0,128))

   exlist1.HotForeColor(WinApi::RGB2int(255,255,255))

   var_Items = exlist1.Items()
    com_Items = var_Items

      com_Items.Add("Item A")

      com_Items.Add("Item B")
```

```
        com_Items.Add("Item C")

    exlist1.EndUpdate()

}
```

VFP

```
with thisform.List1
    .BeginUpdate
    .Columns.Add("Def")
    .HotBackColor = RGB(0,0,128)
    .HotForeColor = RGB(255,255,255)
    with .Items
        .Add("Item A")
        .Add("Item B")
        .Add("Item C")
    endwith
    .EndUpdate
endwith
```

dBASE Plus

```
local oList,var_Items

oList = form.Activex1.nativeObject
oList.BeginUpdate()
oList.Columns.Add("Def")
oList.HotBackColor = 0x800000
oList.HotForeColor = 0xffffff
var_Items = oList.Items
    var_Items.Add("Item A")
    var_Items.Add("Item B")
    var_Items.Add("Item C")
oList.EndUpdate()
```

XBasic (Alpha Five)

```
Dim oList as P
Dim var_Items as P

oList = topparent:CONTROL_ACTIVEX1.activex
oList.BeginUpdate()
oList.Columns.Add("Def")
oList.HotBackColor = 8388608
oList.HotForeColor = 16777215
var_Items = oList.Items
   var_Items.Add("Item A")
   var_Items.Add("Item B")
   var_Items.Add("Item C")
oList.EndUpdate()
```

Delphi 8 (.NET only)

```
with AxList1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := Color.FromArgb(0,0,128);
   HotForeColor := Color.FromArgb(255,255,255);
   with Items do
   begin
      Add('Item A');
      Add('Item B');
      Add('Item C');
   end;
   EndUpdate();
end
```

Delphi (standard)

```
with List1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := RGB(0,0,128);
```

```
   HotForeColor := RGB(255,255,255);
   with Items do
   begin
      Add('Item A');
      Add('Item B');
      Add('Item C');
   end;
   EndUpdate();
end
```

Visual Objects

```
local var_Items as IItems

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:Columns:Add("Def")
oDCOCX_Exontrol1:HotBackColor := RGB(0,0,128)
oDCOCX_Exontrol1:HotForeColor := RGB(255,255,255)
var_Items := oDCOCX_Exontrol1:Items
   var_Items:Add("Item A")
   var_Items:Add("Item B")
   var_Items:Add("Item C")
oDCOCX_Exontrol1:EndUpdate()
```

PowerBuilder

```
OleObject oList,var_Items

oList = ole_1.Object
oList.BeginUpdate()
oList.Columns.Add("Def")
oList.HotBackColor = RGB(0,0,128)
oList.HotForeColor = RGB(255,255,255)
var_Items = oList.Items
   var_Items.Add("Item A")
   var_Items.Add("Item B")
   var_Items.Add("Item C")
oList.EndUpdate()
```

property List.HotForeColor as Color

Retrieves or sets a value that indicates the hot-tracking foreground color.

Type	Description
Color	A color expression that indicates the foreground color for item from the cursor (hovering the item).

By default, the HotForeColor property is 0, which means that the HotForeColor property has no effect. Use the HotForeColor property on a non-zero value to highlight the item from the cursor. The HotBackColor property specifies the background color to highlight the item from the cursor. The ItemFromPoint property gets the item from the cursor. The SelForeColor property specifies the selection foreground color.

The following sample displays a different background color mouse passes over an item.

VBA

```
With List1
   .BeginUpdate
   .Columns.Add "Def"
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .Add "Item A"
      .Add "Item B"
      .Add "Item C"
   End With
   .EndUpdate
End With
```

VB6

```
With List1
   .BeginUpdate
   .Columns.Add "Def"
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .Add "Item A"
```

```
        .Add "Item B"
        .Add "Item C"
    End With
    .EndUpdate
End With
```

VB.NET

```
With Exlist1
    .BeginUpdate()
    .Columns.Add("Def")
    .HotBackColor = Color.FromArgb(0,0,128)
    .HotForeColor = Color.FromArgb(255,255,255)
    With .Items
        .Add("Item A")
        .Add("Item B")
        .Add("Item C")
    End With
    .EndUpdate()
End With
```

VB.NET for /COM

```
With AxList1
    .BeginUpdate()
    .Columns.Add("Def")
    .HotBackColor = RGB(0,0,128)
    .HotForeColor = RGB(255,255,255)
    With .Items
        .Add("Item A")
        .Add("Item B")
        .Add("Item C")
    End With
    .EndUpdate()
End With
```

C++

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

   #import <ExList.dll>
   using namespace EXLISTLib;
*/
EXLISTLib::IListPtr spList1 = GetDlgItem(IDC_LIST1)->GetControlUnknown();
spList1->BeginUpdate();
spList1->GetColumns()->Add(L"Def");
spList1->PutHotBackColor(RGB(0,0,128));
spList1->PutHotForeColor(RGB(255,255,255));
EXLISTLib::IItemsPtr var_Items = spList1->GetItems();
   var_Items->Add("Item A");
   var_Items->Add("Item B");
   var_Items->Add("Item C");
spList1->EndUpdate();
```

C++ Builder

```
List1->BeginUpdate();
List1->Columns->Add(L"Def");
List1->HotBackColor = RGB(0,0,128);
List1->HotForeColor = RGB(255,255,255);
Exlistlib_tlb::IItemsPtr var_Items = List1->Items;
   var_Items->Add(TVariant("Item A"));
   var_Items->Add(TVariant("Item B"));
   var_Items->Add(TVariant("Item C"));
List1->EndUpdate();
```

C#

```
exlist1.BeginUpdate();
exlist1.Columns.Add("Def");
exlist1.HotBackColor = Color.FromArgb(0,0,128);
exlist1.HotForeColor = Color.FromArgb(255,255,255);
exontrol.EXLISTLib.Items var_Items = exlist1.Items;
   var_Items.Add("Item A");
```

```
   var_Items.Add("Item B");
   var_Items.Add("Item C");
exlist1.EndUpdate();
```

JavaScript

```
<OBJECT classid="clsid:CD481F4D-2D25-4759-803F-752C568F53B7" id="List1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   List1.BeginUpdate()

   List1.Columns.Add("Def")

   List1.HotBackColor = 8388608

   List1.HotForeColor = 16777215

   var var_Items = List1.Items

      var_Items.Add("Item A")

      var_Items.Add("Item B")

      var_Items.Add("Item C")

   List1.EndUpdate()

</SCRIPT>
```

C# for /COM

```
axList1.BeginUpdate();
axList1.Columns.Add("Def");
axList1.HotBackColor = Color.FromArgb(0,0,128);
axList1.HotForeColor = Color.FromArgb(255,255,255);
EXLISTLib.Items var_Items = axList1.Items;
   var_Items.Add("Item A");
```

```
    var_Items.Add("Item B");
    var_Items.Add("Item C");
axList1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Items

    anytype var_Items




    super()


    exlist1.BeginUpdate()

    exlist1.Columns().Add("Def")

    exlist1.HotBackColor(WinApi::RGB2int(0,0,128))

    exlist1.HotForeColor(WinApi::RGB2int(255,255,255))

    var_Items = exlist1.Items()
     com_Items = var_Items

        com_Items.Add("Item A")

        com_Items.Add("Item B")

        com_Items.Add("Item C")

    exlist1.EndUpdate()
```

```
}
```

VFP

```
with thisform.List1
   .BeginUpdate
   .Columns.Add("Def")
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   with .Items
      .Add("Item A")
      .Add("Item B")
      .Add("Item C")
   endwith
   .EndUpdate
endwith
```

dBASE Plus

```
local oList,var_Items

oList = form.Activex1.nativeObject
oList.BeginUpdate()
oList.Columns.Add("Def")
oList.HotBackColor = 0x800000
oList.HotForeColor = 0xffffff
var_Items = oList.Items
   var_Items.Add("Item A")
   var_Items.Add("Item B")
   var_Items.Add("Item C")
oList.EndUpdate()
```

XBasic (Alpha Five)

```
Dim oList as P
Dim var_Items as P

oList = topparent:CONTROL_ACTIVEX1.activex
```

```
oList.BeginUpdate()
oList.Columns.Add("Def")
oList.HotBackColor = 8388608
oList.HotForeColor = 16777215
var_Items = oList.Items
   var_Items.Add("Item A")
   var_Items.Add("Item B")
   var_Items.Add("Item C")
oList.EndUpdate()
```

Delphi 8 (.NET only)

```
with AxList1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := Color.FromArgb(0,0,128);
   HotForeColor := Color.FromArgb(255,255,255);
   with Items do
   begin
      Add('Item A');
      Add('Item B');
      Add('Item C');
   end;
   EndUpdate();
end
```

Delphi (standard)

```
with List1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := RGB(0,0,128);
   HotForeColor := RGB(255,255,255);
   with Items do
   begin
      Add('Item A');
```

```
    Add('Item B');
    Add('Item C');
  end;
  EndUpdate();
end
```

Visual Objects

```
local var_Items as IItems

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:Columns:Add("Def")
oDCOCX_Exontrol1:HotBackColor := RGB(0,0,128)
oDCOCX_Exontrol1:HotForeColor := RGB(255,255,255)
var_Items := oDCOCX_Exontrol1:Items
  var_Items:Add("Item A")
  var_Items:Add("Item B")
  var_Items:Add("Item C")
oDCOCX_Exontrol1:EndUpdate()
```

PowerBuilder

```
OleObject oList,var_Items

oList = ole_1.Object
oList.BeginUpdate()
oList.Columns.Add("Def")
oList.HotBackColor = RGB(0,0,128)
oList.HotForeColor = RGB(255,255,255)
var_Items = oList.Items
  var_Items.Add("Item A")
  var_Items.Add("Item B")
  var_Items.Add("Item C")
oList.EndUpdate()
```

property List.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. *If the Key is "OLEDragDropImage", it indicates the picture to be shown while the control performs OLE Drag and Drop operations. Currently, the "OLEDragDropImage" has effect only for /COM version. In other words, you can specify a custom-sized picture rather than image of the dragging items, if you specify a picture with the name "OLEDragDropImage" (no quotes included).*
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>"
```

property List.hWnd as Long

Retrieves the control's window handle.

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

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

property List.HyperLinkColor as Color

Specifies the hyperlink color.

Type	Description
Color	A color expression that defines the color used by hyperlink cells.

Use the HyperLinkColor property to specify the color used when the cursor is over the hyperlink cells. A hyperlink cell has the CellHyperLink property true.

method List.Images (Handle as Variant)

Sets at runtime the control's image list. The Handle should be a handle to an Image List control (HIMAGELIST type).

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 user can add images at design time, by drag and drop files to list's image holder. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ReplaceIcon method to update the images list collection at runtime. Use the CellImage, CellImages properties to assign icons to a cell. Use the CellPicture property to assign a custom size picture to a cell. Use the CheckImage or RadioImage property to specify a different look for checkboxes or radio buttons in the cells.

The following VB sample loads an icon using the BASE64 encoded string:

```
With List1
   .BeginUpdate
      .Images
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

      .Columns.Add("Column 1").HeaderImage = 1
      With .Items
         Dim i As Long
         i = .Add("Item 1")
         .CellImages(i, 0) = "2,3"
      End With
   .EndUpdate
End With
```

The following VB sample uses the Microsoft Image List control:

```
List1.Images ImageList1.hImageList
```

The following C++ sample loads icons from a BASE64 encoded string:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_list.BeginUpdate();
CString s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn
```

```
s +=
"/xoAw9ZiFdxbAAGVxM5yOTzkPy+MzGRpmdx2kl2epGY1WgxmZl+Yyery2yyGHyeirGoo+(

s +=
"NbDDLO2xz5PlBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwyu0UPS/U

s +=
"kSSAAkqUU2nE20gmp5oo6JwH+eZ31EjJwB+eBn1K/AHnBWIfvwAZwACYAHsMy9cIMyFeE

m_list.Images( COleVariant( s ) );
m_list.GetColumns().Add( "Column 1" );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItems items = m_list.GetItems();
long h = items.Add( COleVariant( "Item 1" ) );
items.SetCellImage( h, COleVariant( (long) 0 ), 1 );
h = items.Add( COleVariant( "Item 2" ) );
items.SetCellImages( h, COleVariant( (long) 0 ), COleVariant( "2,3" ) );
m_list.EndUpdate();
```

The following VB.NET sample loads icons from a BASE64 encoded string:

```
Dim s As String
With AxList1
   .BeginUpdate()
   s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

   s = s + "Poyf5xoojKAg"
   .Images(s)

   .Columns.Add("Column 1")
   With .Items
      Dim h As Integer
      h = .Add("Item 1")
      .CellImage(h, 0) = 1
      h = .Add("Item 2")
```

```
        .CellImages(h, 0) = "2,3"
    End With
    .EndUpdate()
End With
```

The following C# sample loads icons from a BASE64 encoded string:

```
axList1.BeginUpdate();
string s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

s = s + "Poyf5xoojKAg";
axList1.Images(s);
axList1.Columns.Add("Column 1");
int h = axList1.Items.Add("Item 1");
axList1.Items.set_CellImage(h, 0, 1);
h = axList1.Items.Add("Item 2");
axList1.Items.set_CellImages(h, 0, "2,3");
axList1.EndUpdate();
```

The following VFP sample loads icons from a BASE64 encoded string:

```
local s
With thisform.List1
    .BeginUpdate()
    s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

    s = s +
"dr1fsFhsVjslls1ntFptVrtltt1vuFxuVzul1u13vF5vV7vl9v1/wGBnqAQEZwmCxFhYGLib/xoAw9

    s = s +
"Goo+03mM02Jzee029y2Ewum2+FnOTlGezHNx0b3/C3U258a4mP5HVvOw52s2fg2vH6ml

    s = s +
"kicAJnCbsNbDDLO2xz5PlBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwy

    s = s +
```

```
"wi/8iNPJMjSo0clvjMLuTHLkJTNCqVTSms2Tkq8jTzOcVP/BsePUocQLDQ9AJ3LtFUbR1Hqaiw

    s = s + "Poyf5xoojKAg"
    .Images(s)

    .Columns.Add("Column 1")
    With .Items
        local i
        i = .Add("Item 1")
        .CellImage(i, 0) = 1
        i = .Add("Item 2")
        .CellImages(i, 0) = "2,3"
    EndWith
    .EndUpdate()
EndWith
```

property List.ImageSize as Long

Retrieves or sets the size of control' icons/images/check-boxes/radio-buttons.

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.

The ImageSize property defines the size to display the following UI elements:

- any icon that a cell or column displays (<img>number</img> ex-html tag, CellImage, CellImages)
- check-box or radio-buttons (CellHasCheckBox, CellHasRadioButton)
- header's sorting or drop down-filter glyphs

property List.ItemFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, ColIndex as Long, HitTestInfo as HitTestInfoEnum) as Long

Retrieves the item from point.

Type	Description
X as OLE_XPOS_PIXELS	A single expression that indicates the X position in client coordinate.
Y as OLE_YPOS_PIXELS	A single expression that indicates the Y position in client coordinate.
ColIndex as Long	A long value that indicates the column's index.
HitTestInfo as HitTestInfoEnum	A HitTestInfoEnum expression that determines on return, the position of the cursor within the cell.
Long	A long expression that indicates item's index from point (X,Y)

Use the ItemFromPoint property to get the item from the point specified by the {X,Y}. The X and Y coordinates are expressed in client coordinates, so a conversion must be done in case your coordinates are relative to the screen or to other window. **If the X parameter is -1 and Y parameter is -1 the ItemFromPoint property determines the index of the item from the cursor**. Use the ColumnFromPoint property to retrieve the column from cursor. Use the SelectableItem property to specify the user can select an item.

The following VB sample displays the cell over the cursor:

```
Private Sub List1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim c As Long, i As Long, hit As HitTestInfoEnum
  With List1
    i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If (i >= 0) Then
      If (c >= 0) Then
        Debug.Print .Items.Caption(i, c)
      End If
    End If
  End With
End Sub
```

The following C++ sample displays the cell over the cursor:

```
void OnMouseMoveList1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
   if ( i >= 0 )
   {
      CItems items = m_list.GetItems();
      CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
      OutputDebugString( strCaption );
   }
}
```

The following VB.NET sample displays the cell over the cursor:

```
Private Sub AxList1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseMoveEvent) Handles AxList1.MouseMoveEvent
   Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
   Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
   If (i >= 0) Then
      With AxList1.Items
         Debug.Write(.Caption(i, c))
      End With
   End If
End Sub
```

The following C# sample displays the cell over the cursor:

```
private void axList1_MouseMoveEvent(object sender,
AxEXLISTLib._IListEvents_MouseMoveEvent e)
{
   EXLISTLib.HitTestInfoEnum hit;
   int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
   if (i >= 0)
   {
      System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(i, c).ToString());
   }
}
```

The following VFP sample displays the cell over the cursor:

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

local c, i, hit
With thisform.List1
   c = 0
   hit = 0
   i = .ItemFromPoint(x, y, @c, @hit)
   If (i >= 0)
      wait window nowait .Items.Caption(i, c)
   EndIf
EndWith
```

property List.Items as Items

Retrieves the control's item collection.

Type	Description
Items	An Items object that holds the control's items collection.

Use the Items property to access the items control. Use the Items collection to add or remove items in the control. Use the PutItems method to load items from a safe array. Use the Add method to add new items to the control's items collection. Use the DataSource to add new columns and items to the control. Adding new items fails if the control has no columns.

property List.ItemsAllowSizing as ItemsAllowSizingEnum

Retrieves or sets a value that indicates whether a user can resize items at run-time.

Type	Description
ItemsAllowSizingEnum	An ItemsAllowSizingEnum expression that specifies whether the user can resize a single item at runtime, or all items, at once.

By default, the ItemsAllowSizing property is exNoSizing. Use the ItemsAllowSizing property to specify whether all items are resizable. Use the ItemAllowSizing property of the Items object to specify only when few items are resizable or not. Use the ItemHeight property to specify the height of the item. The CellSingleLine property specifies whether a cell displays its caption using multiple lines. The DefaultItemHeight property specifies the default height of the items. The DefaultItemHeight property affects only items that are going to be added. It doesn't affect items already added.

property List.Layout as String

Saves or loads the control's layout, such as positions of the columns, scroll position, filtering values.

Type	Description
String	A String expression that specifies the control's layout.

You can use the Layout property to store the control's layout and to restore the layout later. For instance, you can save the control's Layout property to a file when the application is closing, and you can restore the control's layout when the application is loaded. The Layout property saves almost all of the control's properties that user can change at runtime (like changing the column's position by drag and drop). The Layout property does NOT save the control's data, so the Layout property should be called once you loaded the data from your database, xml or any other alternative. Once the data is loaded, you can call the Layout property to restore the View as it was saved. Before closing the application, you can call the Layout property and save the content to a file for reading next time the application is opened.

The Layout property saves/loads the following information:

- columns size and position
- current selection
- scrolling position and size
- sorting columns
- filtering options
- SearchColumnIndex property, indicates the column where the user can use the control's incremental searching.

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.

Generally, the Layout property can be used to save / load the control's layout (or as it is displayed). Thought, you can benefit of this property to sort the control using one or more columns as follows:

- multiplesort="";singlesort="", removes any previously sorting
- multiplesort="C3:1", sorts ascending the column with the index 3 (and add it to the sort bar if visible)
- singlesort="C4:2", sorts descending the column with the index 4 (it is not added to sort bar panel)

- multiplesort="C3:1";singlesort="C4:2", sorts ascending the column with the index 3 (and add it to the sort bar if visible), and sorts descending the column with the index 4. In other words, it re-sort the control by columns 3 and 4.
- multiplesort="C3:1 C5:2";singlesort="C4:2", sorts ascending the column with the index 3 (and add it to the sort bar if visible), sorts descending the column with the index 5 (and add it to the sort bar if visible), and sorts descending the column with the index 4. In other words, it re-sort the control by columns 3, 5 and 4.

The format of the Layout in non-encoded form is like follows:

```
c0.filtertype=0
c0.position=0
c0.select=0
c0.visible=1
c0.width=96
....
columns=13
collapse="0-3 5-63 80-81 83"
filterprompt=""
focus=8
focuscolumnindex=0
hasfilter=1
hscroll=0
multiplesort="C12:1 C2:2"
searchcolumnindex=3
select="39 2 13 8"
selectcolumnindex=0
singlesort="C5:2"
vscroll=12
vscrolloffset=0
```

property List.MarkSearchColumn as Boolean

Retrieves or sets a value that indicates whether the searching column is marked or unmarked

Type	Description
Boolean	A boolean expression that indicates whether the searching column is marked or unmarked.

The control supports incremental search feature. The MarkSearchColumn property specifies whether the control highlights the searching column. Use the SearchColumnIndex property to specify the index of the searching column. The user can change the searching column by pressing the TAB ort Shift + TAB key. Use the AutoSearch property to specify whether the control enables the incremental searching feature. Use the AutoSearch property to specify the type of incremental searching the control supports within the column. Use the UseTabKey property to specify whether the control uses the TAB key.

method List.OLEDrag ()

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

Type	Description

Only for internal use.

property List.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.

In the /NET Assembly, you have to use the AllowDrop property as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

Currently, the control supports only manual OLE Drag and Drop operations. See the OLEStartDrag and OLEDragDrop events for more details about implementing drag and drop operations into the control. Use the Background(exDragDropBefore) property to specify the visual appearance for the dragging items, before painting the items. Use the Background(exDragDropAfter) property to specify the visual appearance for the dragging items, after painting the items. Use the Background(exDragDropList) property to specify the graphic feedback for the item from the cursor, while the OLE drag and drop operation is running. *If the HTMLPicture("OLEDragDropImage") property points to a valid picture object, it indicates the picture to be shown while the control performs OLE Drag and Drop operations. Currently, the "OLEDragDropImage" has effect only for /COM version. In other words, you can specify a custom-sized picture rather than image of the dragging items, if you specify a picture with the name "OLEDragDropImage" (no quotes included).*

property List.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.

By default, the control has no picture associated. The control uses the PictureDisplay property to determine how the picture is displayed on the control's background. Use the PictureLevelHeader property to specify the picture on the control's levels header bar. Use the CellPicture property to assign a picture to a cell. Use the BackColor property to specify the control's background color. Use the SelBackMode property to define how the selected items are painted.

property List.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.

By default, the PictureDisplay property is exTile. Use the PictureDisplay property specifies how the Picture is displayed on the control's background. If the control has no picture associated the PictureDisplay property has no effect. Use the CellPicture property to assign a picture to a cell. Use the BackColor property to specify the control's background color. Use the SelBackMode property to define how the selected items are painted.

property List.PictureDisplayLevelHeader as PictureDisplayEnum

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

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the picture is displayed on the control's header.

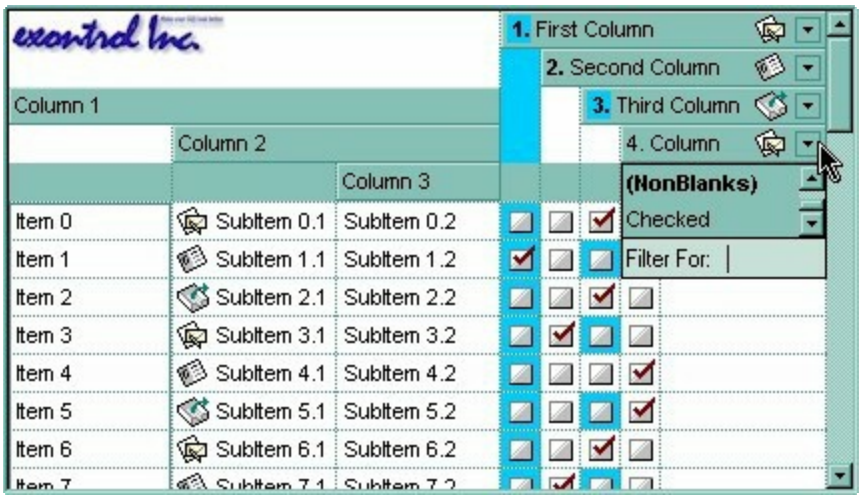
Use the PictureDisplayLevelHeader property to arrange the picture on the control's multiple levels header bar. Use the PictureLevelHeader property to load a picture on the control's header bar when it displays multiple levels. The control's header bar displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the LevelKey property to specify the control's level key.

property List.PictureLevelHeader as IPictureDisp

Retrieves or sets a graphic to be displayed in the control's header when multiple levels is on.

Type	Description
IPictureDisp	A Picture object being displayed on the control's header bar when multiple levels is on.

Use the PictureLevelHeader property to display a picture on the control's header bar when it displays the columns using multiple levels. Use the PictureDisplayLevelHeader property to arrange the picture on the control's multiple levels header bar. The control's header bar displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the LevelKey property to specify the control's level key. Use the Picture property to display a picture on the control's list area. Use the BackColorLevelHeader property to specify the background color for parts of the control's header bar that are not occupied by column's headers.



method List.PutItems (Items as Variant, [Index as Variant])

Adds an array of integer, long, date, string, double, float, or variant arrays to the list, beginning at Index.

Type	Description
Items as Variant	An array that control uses to fill with. The array can be one or two- dimensional. If the array is one-dimensional, the control requires one column being added before calling the PutItems method. If the Items parameter indicates a two-dimensional array, the first dimension defines the columns, while the second defines the number of items to be loaded. For instance, a(2,100) means 2 columns and 100 items.
Index as Variant	Optional. Only for future use.

Use the PutItems property when you have a table of elements stored by an array. Use the GetItems method to get the items collection to a safe array. Use the Items property to access the control's items collection. Use the Add method to add new items to the control's items collection. Use the ColumnAutoResize property to specify whether the visible columns should fit the control's client area. Use the DataSource property to bind the control to an ADO or DAO recordset. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample loads an array to your control:

```
With List1
   .BeginUpdate
   .Columns.Add "Column 1"
   .PutItems Array("Item 1", "Item 2", "Item 3")
   .EndUpdate
End With
```

The following VB sample loads data from a string using the " " as delimiter:

```
Private Sub Form_Load()
   Dim s As String
   s = "a b c d"

   With List1
```

```
        .BeginUpdate
        .Columns.Add "Default"

        .PutItems Split(s, " ")
        .EndUpdate
    End With
End Sub
```

The following VB sample loads an array of strings:

```
Dim v(2, 2) As String
v(0, 0) = "One"
v(0, 1) = "Two"
v(0, 2) = "Three"
v(1, 0) = "One"
v(1, 1) = "Two"
v(1, 2) = "Three"
v(2, 0) = "One"
v(2, 1) = "Two"
v(2, 2) = "Three"

List1.BeginUpdate
    List1.Columns.Add "Column 1"
    List1.Columns.Add "Column 2"
    List1.Columns.Add "Column 3"
    List1.PutItems v
List1.EndUpdate
```

The following VB sample loads an ADO recordset using PutItems method.

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Orders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Opens the table using static mode

List1.BeginUpdate
For Each f In rs.Fields
    List1.Columns.Add f.Name
Next
```

```
List1.PutItems rs.GetRows()
List1.EndUpdate
```

The following C++ sample loads records from an ADO recordset, using the PutItems method:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"


#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;


_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
{
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_list.BeginUpdate();
         m_list.SetColumnAutoResize( FALSE );
         CColumns columns = m_list.GetColumns();
         for ( long i = 0; i < spRecordset->Fields->Count; i++ )
            columns.Add( spRecordset->Fields->GetItem(i)->Name );
         COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
         m_list.PutItems( &spRecordset->GetRows(-1), vtMissing );
         m_list.EndUpdate();
```

```
        }
    }
    catch ( _com_error& e )
    {
        AfxMessageBox( e.Description() );
    }
}
```

The sample uses the #import statement to import ADODB recordset's type library. The sample enumerates the fields in the recordset and adds a new column for each field found. Also, the sample uses the GetRows method of the ADODB recordset to retrieves multiple records of a Recordset object into a safe array. Please consult the ADODB documentation for the GetRows property specification.

The following VB.NET sample loads an array of elements:

```
With AxList1
    .BeginUpdate()
    .Columns.Add("Column 1")
    Dim c() As Integer = {1, 2, 3, 4, 5}
    .PutItems(c)
    .EndUpdate()
End With
```

property List.RadioImage(Checked as Boolean) as Long

Retrieves or sets a value that indicates the index of image used by cells of radio type.

Type	Description
Checked as Boolean	A boolean expression that indicates the radio's state: True - Checked, False - Unchecked.
Long	A long expression that indicates the index of the image used to paint the cells of radio type. The last 7 bits in the high significant byte of the long 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.

Use RadioImage and CheckImage properties to define the icons used for radio and check box cells. The RadioImage property defines the index of the icon being used by radio buttons. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellHasCheckBox property to assign a checkbox to a cell. Use the CellImage or CellImages property to assign one or multiple icons to a cell. Use the CellPicture property to assign a picture to a cell. Use the CellStateChanged event to notify your application when the cell's state is changed. Use the Images method to insert icons at runtime. The following samples require a control with icons, else nothing will be changed. The ImageSize property defines the size (width/height) of the control's check-box/radio-button.

property List.RClickSelect as Boolean

Retrieves or sets a value that indicates whether an item is selected using right mouse button.

Type	Description
Boolean	A boolean expression that indicates whether an item is selected using right mouse button.

Use the RClickSelect property to allow users select items using the right click. By default, the RClickSelect property is False. The control fires the SelectionChanged event when user selects an item. Use the SelectItem property to select programmatically select an item. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the FocusItem property to get the focused item. Use the ItemFromPoint property to retrieve an item from the point.

method List.Refresh ()

Refreshes the control's content.

Type	Description

The Refresh method forces repainting the control. Use the BeginUpdate and EndUpdate methods to maintain the control's performance while adding multiple items or columns. Use the hWnd property to get the handle of the control's window.

The following VB sample calls the Refresh method:

```
List1.Refresh
```

The following C++ sample calls the Refresh method:

```
m_list.Refresh();
```

The following VB.NET sample calls the Refresh method:

```
AxList1.CtlRefresh()
```

In VB.NET the System.Windows.Forms.Control class has already a Refresh method, so the CtlRefresh method should be called.

The following C# sample calls the Refresh method:

```
axList1.CtlRefresh();
```

In C# the System.Windows.Forms.Control class has already a Refresh method, so the CtlRefresh method should be called.

The following VFP sample calls the Refresh method:

```
thisform.List1.Object.Refresh()
```

method List.RemoveSelection ()

Removes the selected items (including the descendents)

Type	Description

The RemoveSelection method removes the selected items. The Remove method removes a specific item. The UnselectAll method unselects all items in the list.

method List.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 = List1.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 = List1.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:

List1.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:

List1.ReplaceIcon 0, -1

property List.RightToLeft as Boolean

Indicates whether the component should draw right-to-left for RTL languages.

Type	Description
Boolean	A boolean expression that specifies whether the control is drawn from right to left or from left to right.

By default, the RightToLeft property is False. The RightToLeft gets or sets a value indicating whether control's elements are aligned to right or left. The RightTolLeft property affects all columns, and future columns being added.

Changing the RightToLeft property on True does the following:

- displays the vertical scroll bar on the left side of the control (Scrollbars property)
- flips the order of the columns (Position property)
- change the column's alignment to right, if the column is not centered (Alignment property, HeaderAlignment property, HeaderImageAlignment property)
- reverse the order of the drawing parts in the cells (Def(exCellDrawPartsOrder) property to "caption,picture,icons,icon,check")
- aligns the locked columns to the right (CountLockedColumns property)
- aligns the control's group-by bar / sort bar to the right (SortBarVisible property)
- the control's filter bar/prompt/close is aligned to the right (FilterBarPromptVisible property)

(By default) Changing the RightToLeft property on False does the following:

- displays the vertical scroll bar on the right side of the control (Scrollbars property)
- flips the order of the columns (Position property)
- change the column's alignment to left, if the column is not centered (Alignment property, HeaderAlignment property, HeaderImageAlignment property)
- reverse the order of the drawing parts in the cells (Def(exCellDrawPartsOrder) property to "check,icon,icons,picture,caption")
- aligns the locked columns to the left (CountLockedColumns property)
- aligns the control's group-by bar / sort bar to the left (SortBarVisible property)
- the control's filter bar/prompt/close is aligned to the left (FilterBarPromptVisible property)

property List.ScrollBars as ScrollBarsEnum

Specifies the type of scroll bars that control has.

Type	Description
ScrollBarsEnum	A ScrollBarsEnum expression that indicates the type of control's scroll bars.

Use the ScrollBars property to disable the control's scroll bars. By default, the ScrollBars property is exBoth, so both scroll bars are used if necessarily. Use the ScrollPos property to get the control's scroll position. Use the EnsureVisibleItem method to ensure that an item fits the control's client area. The EnsureVisibleColumn method ensures that the given column fits the control's client area. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.

property List.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 ScrollBars property to specify which scroll bar is visible or hidden in the control. 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 List.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 ScrollBars property to specify which scroll bar is visible or hidden in the control. 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 List.ScrollBySingleLine as Boolean

Retrieves or sets a value that indicates whether the control scrolls the lines to the end.

Type	Description
Boolean	A boolean expression that indicates whether the control scrolls the content row by row

By default, the ScrollBySingleLine property is False. We recommend to set the ScrollBySingleLine property on True if you have one of the following:

- If you have at least a cell that has CellSingleLine property on exCaptionWordWrap / exCaptionBreakWrap / False, or a column with Def(exCellSingleLine) on exCaptionWordWrap / exCaptionBreakWrap / False
- If the control displays items with different height. Use the ItemHeight property to specify the item's height.

Use the EnsureVisibleItem property to ensure that an item fits the control's client area. Use the ScrollBars property to hide the control's scroll bars. Use the ItemsAllowSizing property to specify whether all items are resizable or not. Use the ItemAllowSizing property to specify whether the user can resize the item at runtime.

property List.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. Use the ScrollBars property to specify the visible scrollbars in the control. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar.

property List.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 ScrollBars property to specify which scroll bar is visible or hidden in the control. 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 List.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.

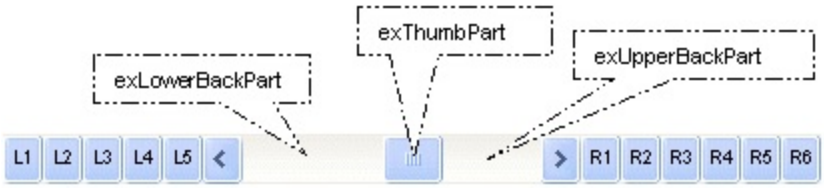
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property List.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. Use the ScrollBars property to specify the visible scrollbars in the control. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. 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 List1
   .BeginUpdate
      .ScrollBars = exDisableBoth
      .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 AxList1
   .BeginUpdate()
   .ScrollBars = EXLISTLib.ScrollBarsEnum.exDisableBoth
   .set_ScrollPartVisible(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part Or EXLISTLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.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 :

```
axList1.BeginUpdate();
axList1.ScrollBars = EXLISTLib.ScrollBarsEnum.exDisableBoth;
axList1.set_ScrollPartVisible(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part | EXLISTLib.ScrollPartEnum.exRightB1Part, true);
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axList1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_list.BeginUpdate();
m_list.SetScrollBars( 15 /*exDisableBoth*/ );
m_list.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32 /*exRightB1Part*/,
TRUE );
m_list.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_list.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2") );
m_list.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.List1
   .BeginUpdate
      .ScrollBars = 15
      .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 List.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.

The following VB sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
With List1
   .ScrollPartCaption(exHScroll,exLowerBackPart) = "left"
   .ScrollPartCaptionAlignment(exHScroll,exLowerBackPart) = LeftAlignment
   .ScrollPartCaption(exHScroll,exUpperBackPart) = "right"
   .ScrollPartCaptionAlignment(exHScroll,exUpperBackPart) = RightAlignment
   .ColumnAutoResize = False
   .Columns.Add 1
   .Columns.Add 2
   .Columns.Add 3
   .Columns.Add 4
End With
```

The following VB.NET sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
With AxList1
```

```
.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollPartEnum.exLower

.set_ScrollPartCaptionAlignment(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollPartEnu

.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollPartEnum.exUppe

.set_ScrollPartCaptionAlignment(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollPartEnu

    .ColumnAutoResize = False
    .Columns.Add 1
    .Columns.Add 2
    .Columns.Add 3
    .Columns.Add 4
End With
```

The following C# sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollPartEnum.e

axList1.set_ScrollPartCaptionAlignment(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollP

axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollPartEnum.e

axList1.set_ScrollPartCaptionAlignment(EXLISTLib.ScrollBarEnum.exHScroll,EXLISTLib.ScrollP

axList1.ColumnAutoResize = false;
axList1.Columns.Add(1.ToString());
axList1.Columns.Add(2.ToString());
axList1.Columns.Add(3.ToString());
axList1.Columns.Add(4.ToString());
```

The following C++ sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's

horizontal scroll bar:

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

   #import "ExList.dll"
   using namespace EXLISTLib;
*/
EXLISTLib::IListPtr spList1 = GetDlgItem(IDC_LIST1)->GetControlUnknown();
spList1->PutScrollPartCaption(EXLISTLib::exHScroll,EXLISTLib::exLowerBackPart,L"left");
spList1-
>PutScrollPartCaptionAlignment(EXLISTLib::exHScroll,EXLISTLib::exLowerBackPart,EXLISTLib

spList1->PutScrollPartCaption(EXLISTLib::exHScroll,EXLISTLib::exUpperBackPart,L"right");
spList1-
>PutScrollPartCaptionAlignment(EXLISTLib::exHScroll,EXLISTLib::exUpperBackPart,EXLISTLib

spList1->PutColumnAutoResize(VARIANT_FALSE);
spList1->GetColumns()->Add(L"1");
spList1->GetColumns()->Add(L"2");
spList1->GetColumns()->Add(L"3");
spList1->GetColumns()->Add(L"4");
```

The following VFP sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
with thisform.List1
   .ScrollPartCaption(1,512) = "left"
   .ScrollPartCaptionAlignment(1,512) = 0
   .ScrollPartCaption(1,128) = "right"
   .ScrollPartCaptionAlignment(1,128) = 2
   .ColumnAutoResize = .F.
   .Columns.Add(1)
   .Columns.Add(2)
   .Columns.Add(3)
   .Columns.Add(4)
```

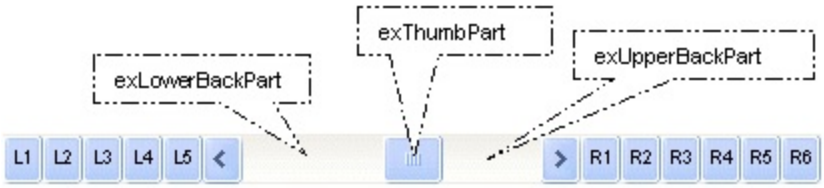
```
endwith
```

property List.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 ScrollBars property to specify the visible scrollbars in the control. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. 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.

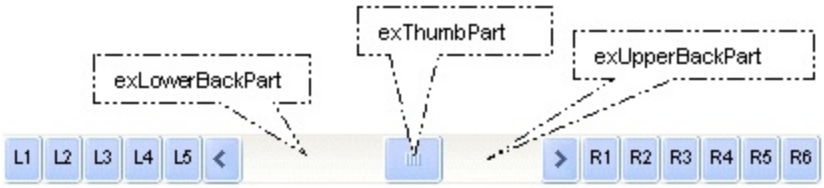


property List.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

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 ScrollBars property to specify the visible scrollbars in the control. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. 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 List1
   .BeginUpdate
      .ScrollBars = exDisableBoth
      .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 AxList1
   .BeginUpdate()
   .ScrollBars = EXLISTLib.ScrollBarsEnum.exDisableBoth
   .set_ScrollPartVisible(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part Or EXLISTLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.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 :

```
axList1.BeginUpdate();
axList1.ScrollBars = EXLISTLib.ScrollBarsEnum.exDisableBoth;
axList1.set_ScrollPartVisible(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part | EXLISTLib.ScrollPartEnum.exRightB1Part, true);
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axList1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_list.BeginUpdate();
m_list.SetScrollBars( 15 /*exDisableBoth*/ );
m_list.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32 /*exRightB1Part*/,
TRUE );
m_list.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_list.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2") );
m_list.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.List1
   .BeginUpdate
      .ScrollBars = 15
      .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 List.ScrollPos(Vertical as Boolean) as Long

Specifies the vertical/horizontal scroll position.

Type	Description
Vertical as Boolean	A boolean expression that specifies the scrollbar being changed. True means Vertical scroll bar, False means Horizontal scroll bar.
Long	A long expression that defines the scroll bar position.

Use the ScrollPos property to change programmatically the position of the control's scroll bar. Use the ScrollPos property to get the horizontal or vertical scroll position. Use the ScrollBars property to define the control's scroll bars.

property List.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 List.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. The OffsetChanged event notifies your application that the user changes the scroll position. Use the SortPartVisible property to specify the parts being visible in the control's scroll bar. Use the ScrollBars property to specify the visible scrollbars in the control.

The following VB sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
Private Sub List1_OffsetChanged(ByVal Horizontal As Boolean, ByVal NewVal As Long)
   If (Not Horizontal) Then
      List1.ScrollToolTip(exVScroll) = "Record " & NewVal
   End If
End Sub
```

The following VB.NET sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
Private Sub AxList1_OffsetChanged(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_OffsetChangedEvent) Handles AxList1.OffsetChanged
   If (Not e.horizontal) Then
      AxList1.set_ScrollToolTip(EXLISTLib.ScrollBarEnum.exVScroll, "Record " &
e.newVal.ToString())
   End If
End Sub
```

The following C++ sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
void OnOffsetChangedList1(BOOL Horizontal, long NewVal)
{
   if ( !Horizontal )
   {
      CString strFormat;
      strFormat.Format( _T("%i"), NewVal );
      m_list.SetScrollToolTip( 0, strFormat );
   }
}
```

The following C# sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
private void axList1_OffsetChanged(object sender,
AxEXLISTLib._IListEvents_OffsetChangedEvent e)
{
   if ( !e.horizontal )
      axList1.set_ScrollToolTip(EXLISTLib.ScrollBarEnum.exVScroll, "Record " +
e.newVal.ToString());
}
```

The following VFP sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
*** ActiveX Control Event ***
LPARAMETERS horizontal, newval

If (1 # horizontal) Then
   thisform.List1.ScrollToolTip(0) = "Record " + ltrim(str(newval))
EndIf
```

property List.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 ScrollBars property to specify which scroll bar is visible or hidden in the control. 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 List.SearchColumnIndex as Long

Retrieves or sets a value indicating the column's index that is used for auto search feature.

Type	Description
Long	A long expression indicating the column's index that is used for auto search feature.

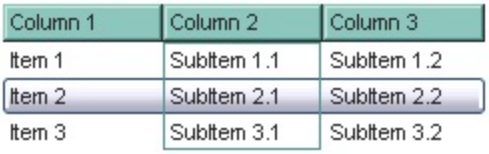
The SearchColumnIndex property indicates the index of the column being used by the control's incremental search feature. The user changes the searching column if he presses TAB or Shift + TAB. Use the UseTabKey property to specify whether the control uses the TAB key. Use the AutoSearch property to specify whether the control enables the incremental searching feature. Use the AutoSearch property to specify the type of incremental searching the control supports within the column. Use the MarkSearchColumn property to hide the rectangle around the searching column.

property List.SelBackColor as Color

Retrieves or sets a value that indicates the selection background color.

Type	Description
Color	A color expression that indicates the color used for painting the selection background. 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.

Use the SelBackColor and SelForeColor properties to define the colors used for selected items. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. The control fires the SelectionChanged event when user changes the selection. Use the SelectableItem property to specify the user can select an item. Use the ShowFocusRect property to specify whether the control marks the focused item. How do I assign a new look for the selected item?



The following VB sample changes the visual appearance for the selected item. 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 item(s):

```
With List1
   With .VisualAppearance
      .Add &H23, App.Path + "\selected.ebn"
   End With
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = &H23000000
End With
```

The sample adds the skin with the index 35 (Hexa 23), and applies to the selected item

using the SelBackColor property.

The following C++ sample applies a new appearance to the selected item(s):

```
#include "Appearance.h"
m_list.GetVisualAppearance().Add( 0x23,
COleVariant(_T("D:\\Temp\\ExList_Help\\selected.ebn")) );
m_list.SetSelBackColor( 0x23000000 );
m_list.SetSelForeColor( 0 );
```

The following VB.NET sample applies a new appearance to the selected item(s):

```
With AxList1
   With .VisualAppearance
      .Add(&H23, "D:\Temp\ExList_Help\selected.ebn")
   End With
   .SelForeColor = Color.Black
   .Template = "SelBackColor = 587202560"
End With
```

The VB.NET sample uses the Template property to assign a new value to the SelBackColor property. The 587202560 value represents &23000000 in hexadecimal.

The following C# sample applies a new appearance to the selected item(s):

```
axList1.VisualAppearance.Add(0x23, "D:\\Temp\\ExList_Help\\selected.ebn");
axList1.Template = "SelBackColor = 587202560";
```

The following VFP sample applies a new appearance to the selected item(s):

```
With thisform.List1
   With .VisualAppearance
      .Add(35, "D:\Temp\ExList_Help\selected.ebn")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = 587202560
EndWith
```

The 587202560 value represents &23000000 in hexadecimal. The 32 value represents &23 in hexadecimal

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 List1
   .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 List1
   .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 List.SelBackMode as BackModeEnum

Retrieves or sets a value that indicates whether the selection is transparent or opaque.

Type	Description
BackModeEnum	A BackModeEnum expression that indicates how the selected items are displayed.

By default, the SelBackMode property is exOpaque. Use the SelBackMode property to specify how the selection is shown in the control. Use the SelBackMode property to specify a specify a semi-transparent color so the selected rows do not lose the colors, pictures, when they are selected. Use the SelBackColor property to specify the visual appearance or the background color for selected items. Use the SelForeColor property to specify the selection foreground color. The SingleSel property specifies whether the control supports single or multiple selection. The control fires the SelectionChanged event when user selects an item. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. The FullRowSelect property specifies whether the full item or a single cell is being selected.

Let's say that you are using the BackColorAlternate property to alternate the colors for rows in the list, so the list with no selection shows as follows:

OrderID	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia
10248	8	8/4/1994	9/1/1994	8/16/1994	3
10249	6	8/5/1994	9/16/1994	8/10/1994	1
10250	2	8/8/1994	9/5/1994	8/12/1994	2
10251	4	8/8/1994	9/5/1994	8/15/1994	1
10252	4	8/9/1994	9/6/1994	8/11/1994	2
10253	3	8/10/1994	8/24/1994	8/16/1994	2
10254	5	8/11/1994	9/8/1994	8/23/1994	2
10255	9	8/12/1994	9/9/1994	8/15/1994	3
10256	3	8/15/1994	9/12/1994	8/17/1994	2
10257	4	8/16/1994	9/13/1994	8/22/1994	3
10258	1	8/17/1994	9/14/1994	8/23/1994	1

By default, the SelBackMode property is **exOpaque**, so the selected items looks like follows:

OrderID	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia
10248	8	8/4/1994	9/1/1994	8/16/1994	3
10249	6	8/5/1994	9/16/1994	8/10/1994	1
10250	2	8/8/1994	9/5/1994	8/12/1994	2
10251	4	8/8/1994	9/5/1994	8/15/1994	1
10252	4	8/9/1994	9/6/1994	8/11/1994	2
10253	3	8/10/1994	8/24/1994	8/16/1994	2
10254	5	8/11/1994	9/8/1994	8/23/1994	2
10255	9	8/12/1994	9/9/1994	8/15/1994	3
10256	3	8/15/1994	9/12/1994	8/17/1994	2
10257	4	8/16/1994	9/13/1994	8/22/1994	3
10258	1	8/17/1994	9/14/1994	8/23/1994	1

And if the SelBackMode property is set on **exTransparent**, the selected rows do not change their background/foreground colors as shown bellow:

OrderID	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia
10248	8	8/4/1994	9/1/1994	8/16/1994	3
10249	6	8/5/1994	9/16/1994	8/10/1994	1
10250	2	8/8/1994	9/5/1994	8/12/1994	2
10251	4	8/8/1994	9/5/1994	8/15/1994	1
10252	4	8/9/1994	9/6/1994	8/11/1994	2
10253	3	8/10/1994	8/24/1994	8/16/1994	2
10254	5	8/11/1994	9/8/1994	8/23/1994	2
10255	9	8/12/1994	9/9/1994	8/15/1994	3
10256	3	8/15/1994	9/12/1994	8/17/1994	2
10257	4	8/16/1994	9/13/1994	8/22/1994	3
10258	1	8/17/1994	9/14/1994	8/23/1994	1

The following screen shot shows the control when no items are selected:

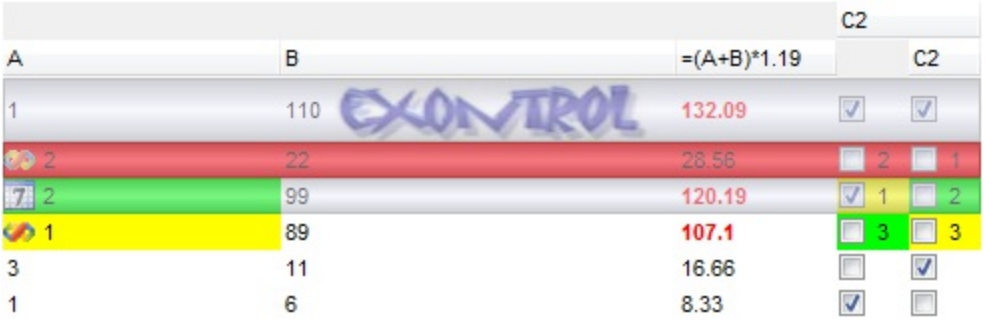
			C2	
A	B	=(A+B)*1.19		C2
1	110	132.09	☑	☑
2	22	28.56	☐ 2	☐ 1
2	99	120.19	☑ 1	☐ 2
1	89	107.1	☐ 3	☐ 3
3	11	16.66	☐	☑
1	6	8.33	☑	☐

The following screen shot shows the first three items selected, while the SelBackMode property is **exOpaque**:

			C2	
A	B	=(A+B)*1.19		C2
1	110	132.09	☑	☑
2	22	28.56	☐ 2	☐ 1
2	99	120.19	☑ 1	☐ 2
1	89	107.1	☐ 3	☐ 3
3	11	16.66	☐	☑
1	6	8.33	☑	☐

The following screen shot shows the first three items selected, while the SelBackMode property is **exOpaque**, and FullRowSelect property is 0:

The following screen shot shows the first three items selected, while the SelBackMode property is **exTransparent**:



The following screen shot shows the first three items selected, while the SelBackMode property is **exGrid**:

property List.SelectColumnIndex as Long

Retrieves or sets a value that indicates control column's index where the user is able to select an item.

Type	Description
Long	A boolean expression that specifies whether the user selects cells only in SelectColumnIndex column, while the FullRowSelect property is False

By default, the SelectColumn property is False. The SelectColumn property has effect only if the FullRowSelect is False. The control displays the selected cell in the SelectColumnIndex column. The SelectColumnIndex property specifies the index of selected column. Use the SelectableItem property to specify the user can select an item.

property List.SelectOnRelease as Boolean

Indicates whether the selection occurs when the user releases the mouse button.

Type	Description
Boolean	A Boolean expression that indicates whether the selection occurs when the user releases the mouse button.

By default, the SelectOnRelease property is False. By default, the selection occurs, as soon as the user clicks an object. The SelectOnRelease property indicates whether the selection occurs when the user releases the mouse button.

property List.SelForeColor as Color

Retrieves or sets a value that indicates the selection foreground color.

Type	Description
Color	A color expression that indicates the selection foreground color.

Use the SelForeColor and SelBackColor properties to change the colors used for selected items. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. The control fires the SelectionChanged event when user changes the selection. Use the SelectableItem property to specify the user can select an item.

property List.SelLength as Long

Returns or sets the number of characters selected.

Type	Description
Long	A long expression that indicates the number of characters selected.

By default, the SelLenght property is -1 (all text gets selected). Use the SelLenght property to specify the number of characters being selected when the edit operations begins. The SelStart and SelLength properties have effect only if the control is editable. Use the AllowEdit property to allow control edits the data using a text box field. Use the Edit method to programmatically edit a cell using a textbox field. The SelLength property must be set in the code, before starting editing the cell. The control fires the BeforeCellEdit event when the control is about to open the text box editor. The control fires the AfterCellEdit property when the edit ends.

property List.SelStart as Long

Returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.

Type	Description
Long	A long expression that indicates the starting point of text selected

By default, the SelStart property is 0 (the text gets selected from the first character). Use the SelStart property to specify the starting point of selected text, when edit operations begins. The SelStart and SelLength properties are valid only if the control is editable. Use the AllowEdit property to allow control edits the data using a text box field. Use the Edit method to programmatically edit a cell using a textbox field. The SelStart property must be set in the code, before starting editing the cell. The control fires the BeforeCellEdit event when the control is about to open the text box editor. The control fires the AfterCellEdit property when the edit ends.

property List.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 indicates whether the marker for the focused cell is visible or hidden.

Use the ShowFocusRect property to hide the rectangle drawn around the focused item. The FocusItem property specifies the handle of the focused item. If there is no focused item the FocusItem property retrieves 0. At one moment, only one item can be focused. When the selection is changed the focused item is changed too. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same.

property List.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,

By default, the ShowImageList property is True. Use the ShowImageList property to hide the control's images list window. The control's images list window is visible only at design time. Use the Images method to associate an images list control to the tree control. Use the RepaceIcon method to add, remove or clear icons in the control's images collection. Use the CellImage, CellImages properties to assign icons to a cell. Use the CellPicture property to assign a picture to a cell. Use the CheckImage or RadioImage property to specify a different look for checkboxes or radio buttons in the cells.

method List.ShowToolTip (ToolTip as String, [Title as Variant], [Alignment as Variant], [X as Variant], [Y as Variant])

Shows the specified tooltip at given position.

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

X as Variant	Specifies the horizontal position to display the tooltip as one of the following: • missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored) • -1, indicates the current horizontal position of the cursor (current x-position) • a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position) • a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)
Y as Variant	Specifies the vertical position to display the tooltip as one of the following: • missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored) • -1, indicates the current vertical position of the cursor (current y-position) • a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position) • a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the MouseMove event. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to change the tooltip's font. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color.

For instance:

- ShowToolTip(`<null>`,`<null>`,,`+8`,`+8`), shows the tooltip of the object moved relative

to its default position

- ShowToolTip(`<null>`,`new title`), adds, changes or replaces the title of the object's tooltip
- ShowToolTip(`new content`), adds, changes or replaces the object's tooltip
- ShowToolTip(`new content`,`new title`), shows the tooltip and title at current position
- ShowToolTip(`new content`,`new title`,,`+8`,`+8`), shows the tooltip and title moved relative to the current position
- ShowToolTip(`new content`,``,,128,128), displays the tooltip at a fixed position
- ShowToolTip(``,``), hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

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

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

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

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

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

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

  gradient-center

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

  outlined

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

  shadow

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

  outline anti-aliasing

property List.SingleSel as Boolean

Retrieves or sets a value that indicates 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 enable multiple selection. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. The control fires the SelectionChanged event when user selects an item. Use the SelForeColor and SelBackColor properties to specify colors for selected items. Use the SelectableItem property to specify the user can select an item. The FullRowSelect property specifies whether the selection spans the entire width of the control.

property List.SingleSort as Boolean

Returns or sets a value that indicates whether the control supports sorting by single or multiple columns.

Type	Description
Boolean	A boolean expression that indicates whether the control supports single or multiple selection.

Use the SingleSort property to allow sorting by multiple columns. Sorting by a single column in the control is a simple matter of clicking on the column head. Sorting by multiple columns, however, is not so obvious. But it's actually quite easy. The user has two options to sort by multiple columns:

- First, sort by the first criterion, by clicking on the column head. Then hold the SHIFT key down as you click on a second heading.
- Click the column head and drag to the control's sort bar in the desired position.

By default, the SingleSort property is True, and so the user can have sorting by a single column only. Use the SortBarVisible property to show the control's sort bar. The SingleSort property is automatically set on False, if the SortBarVisible property is set to True. Use the SortOnClick property to specify the action that control should execute when the user clicks the control's header. Use the SortOrder property to sort a column programmatically. Use the SortPosition property to specify the position of the column in the sorted columns list. The control fires the Sort event when the user sorts a column. Use the ItemBySortPosition property to get the columns being sorted in their order.

For instance, if the control contains multiple sorted columns, changing the SingleSort property on True, erases all the columns in the sorting columns collection, and so no column is sorted.

property List.SortBarCaption as String

Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns.

Type	Description
String	A String expression that indicates the caption of the control's sort bar.

The SortBarCaption property specifies the caption of the control's sort bar, when it contains no sorted columns. Use the SortBarVisible property to show the control's sort bar. Use the BackColorSortBar, BackColorSortBarCaption and ForeColorSortBar properties to specify colors for the control's sort bar. Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarColumnWidth property to specify the width of the column in the control's sort bar. By default, the SortBarCaption property is "Drag a column header here to sort by that column.". Use the Font property to specify the control's font. Use the ItemBySortPosition property to access the columns in the control's sort bar.

The SortBarCaption property may include built-in HTML tags 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.

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

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

Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

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

loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.

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

gradient-center

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

outlined

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

shadow

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

property List.SortBarColumnWidth as Long

Specifies the maximum width a column can be in the control's sort bar.

Type	Description
Long	A long expression that indicates the width of the columns in the control's sort bar. If the value is negative, all columns in the sort bar are displayed with the same width (the absolute value represents the width of the columns, in pixels). If the value is positive, it indicates the maximum width, so the width of the columns in the sort bar may differ.

Use the SortBarColumnWidth property to specify the width of the column in the control's sort bar. Use the SortBarVisible property to show the control's sort bar. Use the Width property to specify the width of the column in the control's header bar. Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarCaption property to specify the caption being displayed in the control's sort bar when it contains no columns.

property List.SortBarHeight as Long

Retrieves or sets a value that indicates the height of the control's sort bar.

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

Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarVisible property to show the control's sort bar. By default, the SortBarHeight property is 18 pixels. Use the HeaderHeight property to specify the height of the control's header bar. Use the SortBarColumnWidth property to specify the width of the columns being displayed in the control's sort bar. Use the BackColorSortBar, BackColorSortBarCaption and ForeColorSortBar properties to specify colors for the control's sort bar. Use the SortBarCaption property to specify the caption being displayed in the control's sort bar when it contains no columns.

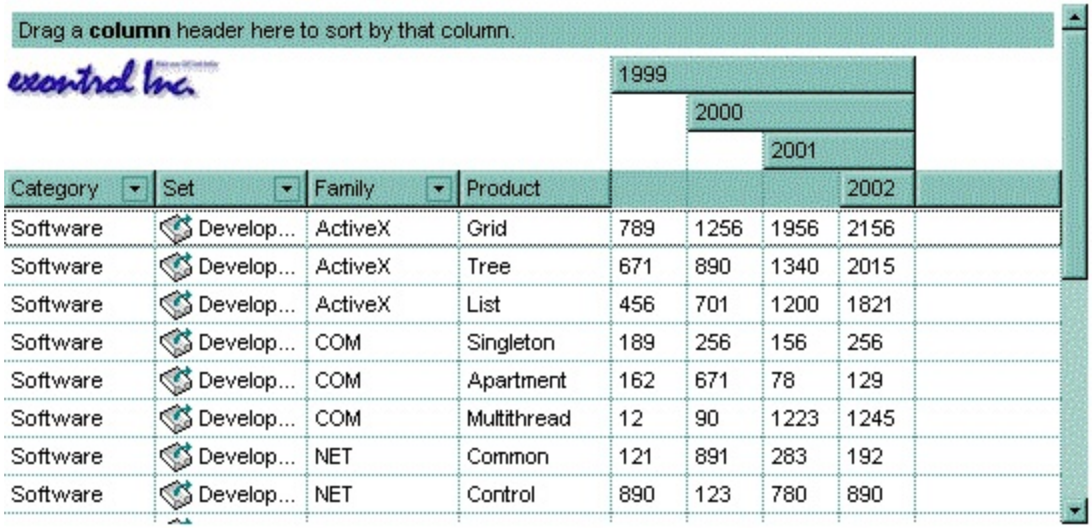
property List.SortBarVisible as Boolean

Retrieves or sets a value that indicates whether control's sort bar is visible or hidden.

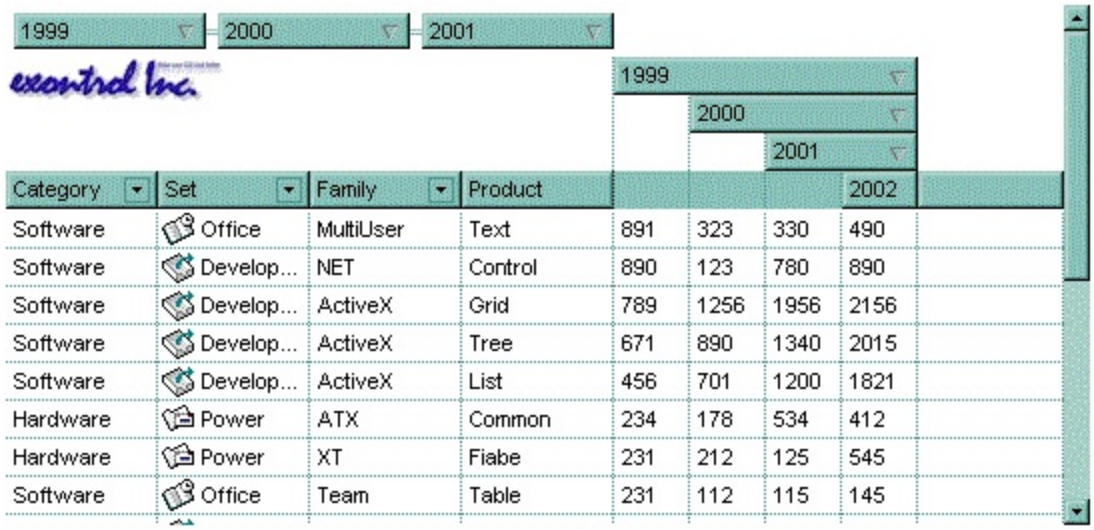
Type	Description
Boolean	A boolean expression that indicates whether the sort bar is visible or hidden.

Use the SortBarVisible property to show the control's sort bar. By default, the SortBarVisible property is False. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. Sorting by a single column in the control is a simple matter of clicking on the column head. Sorting by multiple columns, however, is not so obvious. But it's actually quite easy. The user has two options to sort by multiple columns:

The control's sort bar displays the SortBarCaption expression, when it contains no columns, like follows (the "Drag a **column** header ..." area is the control's sort bar) :



The sort bar displays the list of columns being sorted in their order as follows:



The SortOrder property adds or removes programmatically columns in the control's sort bar. Use the SortPosition property to specify the position of the column in the sorting columns collection. Use the ItemBySortPosition property to access the columns being sorted. Use the SortOnClick property to specify the action that control should execute when user clicks the column's header. Use the AllowSort property to specify whether the user sorts a column by clicking the column's header. The control fires the Sort event when the user sorts a column.

property List.SortOnClick as SortOnClickEnum

Retrieves or sets a value that indicates whether the control sorts automatically the data when the user click on column's caption.

Type	Description
SortOnClickEnum	A SortOnClickEnum expression that indicates the action that control takes when user clicks the column's header.

Use the SortOnClick property to disable sorting items when the user clicks on the column's header. Use the SortBarVisible property to show the control's sort bar. Use the SingleSort property to allow sorting by single or multiple columns. Use the AllowSort property to avoid sorting a column when user clicks the column. Use the DefaultSortOrder property to specify the column's default sort order, when the user first clicks the column's header.

There are two methods to get the items sorted like follows:

- Using the SortOrder property of the Column object::

  ```
  List1.Columns(ColIndex).SortOrder = SortAscending
  ```

  The SortOrder property adds the sorting icon to the column's header, if the DisplaySortIcon property is True.

- Using the Sort method of the Items collection. The Sort sorts the items. The following sample sorts descending the list of items on the "Column 1".

  ```
  List1.Items.Sort "Column 1", False
  ```

The control fires the Sort event when the control sorts a column (the user clicks the column's head) or when the sorting position is changed in the control's sort bar. Use the Sort event to sort the data when the SortOnClk property is exUserSort

property List.Statistics as String

Gives statistics data of objects being hold by the control.

Type	Description
String	A String expression that gives information about objects being loaded into the control.

The Statistics property gives statistics data of objects being hold by the control. The Statistics property gives a rough idea on how many columns, items, cell, bars, links, notes and so on are loaded into the control. Also, the Statistics property gives percentage usage of base-memory of different objects within the memory.

The following output shows how the Statistics looks like, on a 32-bits machine:

```
Cells: 832 x 55 = 45,760 (59.09%)
Control: 1 x 18,568 = 18,568 (23.98%)
Column: 13 x 648 = 8,424 (10.88%)
Item: 64 x 66 = 4,224 (5.45%)
Items: 1 x 372 = 372 (0.48%)
Columns: 1 x 68 = 68 (0.09%)
Appearances: 1 x 28 = 28 (0.04%)
Appearance: 0 x 712 = 0 (0.00%)
CComVariant: 0 x 16 = 0 (0.00%)
CSmartVariant: 0 x 9 = 0 (0.00%)
```

The following output shows how the Statistics looks like, on a 64-bits machine:

```
Cells: 832 x 91 = 75,712 (59.00%)
Control: 1 x 30,208 = 30,208 (23.54%)
Column: 13 x 1,056 = 13,728 (10.70%)
Item: 64 x 122 = 7,808 (6.08%)
Items: 1 x 680 = 680 (0.53%)
Columns: 1 x 136 = 136 (0.11%)
Appearances: 1 x 48 = 48 (0.04%)
Appearance: 0 x 1,168 = 0 (0.00%)
CComVariant: 0 x 24 = 0 (0.00%)
CSmartVariant: 0 x 9 = 0 (0.00%)
```

property List.Template as String

Specifies the control's template.

Type	Description
String	A string expression that defines the control's template

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the ExecuteTemplate property to get the result after executing a script template.

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

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

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

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

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values*

separated by commas. (Sample: h = InsertItem(0,"New Child"))

- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier. For instance, the following code creates an ADOR.Recordset and pass it to the control using the DataSource property:*

```
Dim rs
ColumnAutoResize = False
rs = CreateObject("ADOR.Recordset")
{
Open("Orders","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExList\Sample\SAMPLE.MDB", 3, 3 )
}
DataSource = rs
```

property List.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

```
with (Control)
   TemplateDef = [Dim var_Column]
   TemplateDef = var_Column
   Template = [var_Column.Def(4) = 255]
endwith
```

This sample allocates a variable var_Column, assigns the value to the variable (the second call of the TemplateDef), and the Template call uses the var_Column variable (as an object), to call its Def property with the parameter 4.

Let's say we need to define the background color for a specified column, so we need to call the Def(exCellBackColor) property of the column, to define the color for all cells in the column.

The following **VB6** sample shows setting the Def property such as:

```
With Control
   .Columns.Add("Column 1").Def(exCellBackColor) = 255
   .Columns.Add "Column 2"
   .Items.AddItem 0
   .Items.AddItem 1
```

```
    .Items.AddItem 2
End With
```

In **dBASE Plus**, calling the Def(4) has no effect, instead using the TemplateDef helps you to use properly the Def property as follows:

```
local Control,var_Column

Control = form.Activex1.nativeObject
// Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
with (Control)
    TemplateDef = [Dim var_Column]
    TemplateDef = var_Column
    Template = [var_Column.Def(4) = 255]
endwith
Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

The equivalent sample for **XBasic in A5**, is as follows:

```
Dim Control as P
Dim var_Column as P

Control = topparent:CONTROL_ACTIVEX1.activex
' Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
Control.TemplateDef = "Dim var_Column"
Control.TemplateDef = var_Column
Control.Template = "var_Column.Def(4) = 255"

Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

The samples just call the Column.Def(4) = Value, using the TemplateDef. The first call of TemplateDef property is "Dim var_Column", which indicates that the next call of the TemplateDef will defines the value of the variable var_Column, in other words, it defines the object var_Column. The last call of the Template property uses the var_Column member to use the x-script and so to set the Def property so a new color is being assigned to the column.

The TemplateDef, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or ` characters. If using the ` character, please

make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

method List.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
NewVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

The TemplateDef, TemplatePut, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the*

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property List.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 CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.

property List.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. Use the CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.

property List.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.

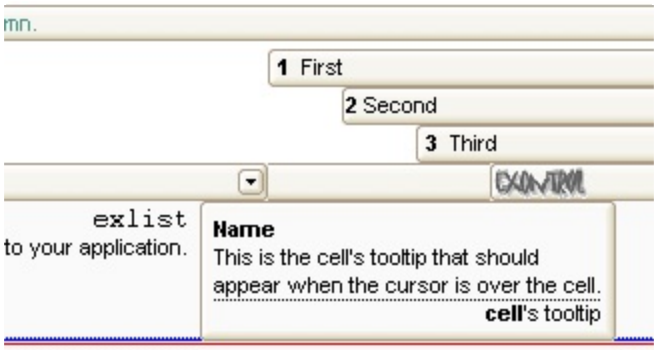
If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to assign a font for 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 CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.

property List.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 CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.



property List.UnboundHandler as IUnboundHandler

Specifies the control's unbound handler.

Type	Description
IUnboundHandler	An object that implements IUnboundHandler notification interface

The control supports unbound mode. In unbound mode, user is responsible for retrieving items. In order to let the control works in unbound mode, the user has to implement the IUnboundHandler notification interface. Use the VirtualMode property to run the control in virtual mode. Use the RemoveAll method to remove all items from the control, after setting the UnboundHandler property to nothing. Use the BeginUpdate and EndUpdate methods, or Refresh method after setting the UnboundHandler property, to reflect the changes in the control's client area.

If the VirtualMode property is True and the DataSource property is set (not empty), the control provides an internal object that implements the IUnboundHandler interface to provide data for the control from the data source.

The following VB sample shows how to activate the control's unbound mode

```
Implements IUnboundHandler
Dim its As Items

Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = 150000
End Property

Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object)
   its.Caption(Index, 0) = Index
   its.CellImage(Index, 0) = Index Mod 2 + 1
End Sub

Private Sub Form_Load()
   With List1
      .ColumnAutoResize = True
      .SortOnClick = False
      .MarkSearchColumn = False
      .Columns.Add "Unbound Column"
```

```
        Set its = .Items
        Set .UnboundHandler = Me

    End With
End Sub
```

property List.UseTabKey as Boolean

Retrieves or sets a value indicating whether the control uses tab key for changing the searching column.

Type	Description
Boolean	A boolean expression indicating whether the control uses tab key for changing the searching column.

By default, the UseTabKey property is True. The UseTabKey property specifies whether the control uses the TAB key to change the searching column. The SearchColumnIndex property indicates the index of the searching column. If the UseTabKey property is False, the TAB key is used to navigate through the form's controls.

property List.UseVisualTheme as UIVisualThemeEnum

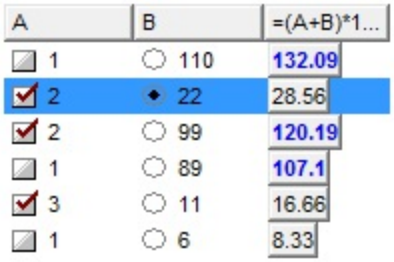
Specifies whether the control uses the current visual theme to display certain UI parts.

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:



property List.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 List.VirtualMode as Boolean

Retrieves or sets a value that indicates whether the control runs in the virtual mode.

Type	Description
Boolean	A boolean expression that indicates whether the control runs in the virtual mode.

Generally, the user needs to run the control in virtual mode, if a table with large number of records needs to be displayed. In virtual mode, the control displays maximum 2,147,483,647 records. **The control is running in virtual mode, only if VirtualMode property is True, and the UnboundHandler property refers an object that implements the IUnboundHandler interface.** Implementing the IUnboundHandler interface is easy because it has only two methods. The first one, ItemsCount specifies the number of records that user needs to display in the control. The second method is ReadItem and it provides data for a specific record. When control is running in the virtual mode, the control loads **only** the items that need to be displayed. If the control is running in the unbound mode (the VirtualMode property is False), the control allocates memory for all records that need to be loaded. The data for each record is loaded only when it is required. The virtual mode has few disadvantages like: the sorting is not available (the user needs to provide sorting data), the control's filtering items is not available, the user cannot add items manually, and so on. The main advantage of the virtual mode is that the control can display large number of records. The unbound mode requires a lot of memory, depending on number of loaded records, but it allows almost all features of the control, including sorting, filtering and so on. Use the ItemToVirtual property to convert the index of the item in the list to the index of the virtual item. Use the VirtualToItem property to get the index of the item in the list giving the index of the virtual item. It is important to know, that the VirtualToItem property ensures that the virtual item fits the control's client area.

Displaying a table, using the Virtual Mode

When you need to display large number of records, you need to provide an object that implements the IUnboundHandler interface. The object provides the number of records that need to be displayed, and data for each record. The VirtualMode property needs to be set on true, and the object you have written needs to be passed to the UnboundHandler property.

The following sample adds a column, and 100 records. The index of each item in the list is displayed.

- Create a new project (Project1)
- Add a control to the form (List1)
- Create a new class module (Class1) and add it to the project
- Open the code of the class, and type "Implements IUnboundHandler"

- Add the handler for the IUnboundHandler_ItemsCount property like follows:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = 100
End Property
```

The control calls the IUnboundHandler_ItemsCount property when the UnboundHandler property is set, to update the vertical scroll range.

- Add the handler for the IUnboundHandler_ReadItem method like follows:

```
 Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object)
   With Source.Items
      .Caption(.VirtualToItem(Index), 0) = Index + 1
   End With
End Sub
```

The control calls the IUnboundHandler_ReadItem method each time when a virtual item becomes visible. Important to notice is that the Items.VirtualToItem property is used to convert the index of the virtual item to the index of the item in the list.

- Open the form's code and add handler for the Form_Load event like follows:

```
Private Sub Form_Load()
   With List1
      .BeginUpdate
         .Columns.Add "Column 1"

         .VirtualMode = True
         Set .UnboundHandler = New Class1
      .EndUpdate
   End With
End Sub
```

- Save the project
- Run the project

The sample runs the control in the virtual mode. The control calls the IUnboundHandler_ItemsCount property when UnboundHandler property is set. The

IUnboundHandler_ReadItem method is invoked when a record needs to be displayed.

Now, that you got the idea of the virtual mode, let's start to complicate the things. Let's suppose that we have a table and we need to display its records in the control.

- Create a new project (Project1)
- Add a control to the form (List1)
- Create a new class module (Class1) and add it to the project
- Add a new variable rs, of Object type like: Public rs as Object. In the following sample, the rs variable holds a reference to an ADO.Recordset object
- Add a new procedure AttachTable like follows:

```
Public Sub AttachTable(ByVal strTable As String, ByVal strPath As String, ByVal g
As EXLISTLibCtl.List)
   Set rs = CreateObject("ADODB.Recordset")
   rs.Open strTable, "Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " & strPath,
3, 3
   With g
      .BeginUpdate
         With .Columns
            Dim f As Variant
            For Each f In rs.Fields
               .Add f.Name
            Next
         End With
      .EndUpdate
   End With
End Sub
```

The AttachTable subroutine opens a table using ADO, and insert in the control's Columns collection a new column for each field found in the table.

- Type "Implements IUnboundHandler" at the beginning of the class
- Implement the IUnboundHandler_ItemsCount property like follows:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As
Long
   IUnboundHandler_ItemsCount = rs.RecordCount
End Property
```

In this case the IUnboundHandler_ItemsCount property the number of records in the

table.

- Implement the IUnboundHandler_ReadItem method like follows:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As
Object)
   rs.Move Index, 1
   Dim i As Long, l As Long
   With Source.Items
      i = 0
      l = .VirtualToItem(Index)
      Dim f As Variant
      For Each f In rs.Fields
         .Caption(l, i) = f.Value
         i = i + 1
      Next
   End With
End Sub
```

The IUnboundHandler_ReadItem method moves the current record using the rs.Move method, at the record with the specified index, and loads values for each cell in the item. If you need to apply colors, font attributes, ... to the items in the control, your handler may change the CellBold, CellForeColor, ... properties.

- Open the form's code, and add a new variable n like: Dim n As New Class1
- Add a handler for the Form_Load event like follows:

```
Private Sub Form_Load()
   With List1
      .BeginUpdate

         n.AttachTable "Select * from Orders",
"D:\Exontrol\ExList\sample\sample.mdb", List1

         .VirtualMode = True
         Set .UnboundHandler = n

      .EndUpdate
   End With
```

```
End Sub
```

The AttachTable method opens the table, and fills the control's Columns collection. The AttachTable method needs to be called before putting the control on virtual mode, because properties of the rs object are called in the ItemsCount and ReadItem methods.

- Save the project
- Run the project

Adding a custom column, when the control is running in the virtual mode.

Let's suppose that we want to display a column with the current position for each record in the table. In this case, we need to add a new column, and we need to change the ReadItem method like follows:

- The Form_Load event should look like:

```
Private Sub Form_Load()
   With List1
      .BeginUpdate

         n.AttachTable "Select * from Orders",
"D:\Exontrol\ExList\sample\sample.mdb", List1

         .VirtualMode = True
         Set .UnboundHandler = n

           With .Columns.Add("Position")
              .Position = 0
           End With

      .EndUpdate
   End With
End Sub
```

- The IUnboundHandler_ReadItem method looks like following:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As
Object)
```

```
    rs.Move Index, 1
    Dim i As Long, l As Long
    With Source.Items
        i = 0
        l = .VirtualToItem(Index)
        Dim f As Variant
        For Each f In rs.Fields
            .Caption(l, i) = f.Value
            i = i + 1
        Next
        .Caption(l, "Position") = Index + 1
    End With
End Sub
```

For instance, if you need to have a column that computes its value based on the other columns, it can be done like this:

```
.Caption(l, "Column") = .Caption(l, "Quantity") * .Caption(l, "UnitPrice")
```

Editing a table using the Virtual Mode

- Locate the Form_Load event and change the AllowEdit property like follows:

```
Private Sub Form_Load()
    With List1
        .BeginUpdate

            .AllowEdit = True

            n.AttachTable "Select * from Orders",
"D:\Exontrol\ExList\sample\sample.mdb", List1

            .VirtualMode = True
            Set .UnboundHandler = n

            With .Columns.Add("Position")
                .Position = 0
            End With
```

```
        .EndUpdate
    End With
End Sub
```

- Add handler for AfterCellEdit event like follows:

```
Private Sub List1_AfterCellEdit(ByVal Index As Long, ByVal ColIndex As Long,
ByVal newCaption As String)
    With List1.Items
        n.Change newCaption, .ItemToVirtual(Index), ColIndex
    End With
End Sub
```

Important to notice is that the Items.ItemToVirtual property is called to convert the index of item in the list to the index of the virtual item being changed. The Change method in the Class1 changes the value in the recordset.

- Add a Change method to the Class1 like follows:

```
Public Sub Change(ByVal newCaption As Variant, ByVal Index As Long, ByVal
ColIndex As Long)
    rs.Move Index, 1
    rs(ColIndex) = newCaption
    rs.Update
End Sub
```

The Change method moves the current position to the Index position in the recordset, and updates the recordset.

- Save and Run the project

Loading a table using the Virtual Mode in C++

The following tutorial will show how to run the control in virtual mode. The sample is a simple MFC dialog based application. Anyway, if your application is different than a MFC dialog based, the base things you need are here, so please find that the following information is useful.

- Create a new project using MFC AppWizard (exe) (ADOVirtual)
- Select Dialog based, for the type of the application
- Insert the control to the application's main dialog (Insert ActiveX Control)

- Save the Project
- Open the MFC Class Wizard, by pressing CTRL + W
- Add a new member variable for IDC_LIST1 resource called m_list. In the meanwhile, please notice that the wizard will ask you 'The ActiveX Control "ExList ActiveX Control" has not been inserted into the project. Developer Studio will do this now and generate a C++ wrapper class for it', and you need to click ok, by following the steps that wizard will ask you to do in order to insert the C++ wrapper classes. (CList1, CItems, CColumns, CColumn, COleFont, CPicture)
- Save the Project
- Open the Dialog Properties, and click the "Clip siblings" and "Clip children"
- Add a new MFC based class, CUnboundHandler derived from the CCmdTarget. We define the CUnboundHandler class to implement the IUnboundHandler interface.
- Import the control's definition using the #import directive, to the CUnboundHandler class like follows:

```
#import "c:\winnt\system32\exlist.dll" rename( "GetItems", "exGetItems" )
```

The #import directive is used to incorporate information from a type library. The content of the type library is converted into C++ classes, mostly describing the COM interfaces. The path to the file need to be changed if the dll is somewhere else. After building the project, the environment generates a namespace EXLISTLib. The generated namespace includes definition for IUnboundHandler interface. It can be accessed using the declaration EXLISTLib::IUnboundHandler

- By default, the destructor of the CUnboundHandler class is declared as protected. The destructor needs to be declared as public (Remove the protected keyword before ~CUnboundHandler).
- Implementing the IUnboundHandler interface using the DECLARE_INTERFACE_MAP, BEGIN_INTERFACE_PART and END_INTERFACE_PART macros. The following snippet needs to be inserted in the class definition like

```
DECLARE_INTERFACE_MAP()

public:
   BEGIN_INTERFACE_PART(Handler, EXLISTLib::IUnboundHandler)
            STDMETHOD(get_ItemsCount)(IDispatch * Source,long* pVal);
      STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source);
   END_INTERFACE_PART(Handler)
```

The CUnboundHandler class definition should look like follows (we have removed the comments added by the wizard):

```
#import "c:\winnt\system32\exlist.dll"  rename( "GetItems", "exGetItems" )

class CUnboundHandler : public CCmdTarget
{
    DECLARE_DYNCREATE(CUnboundHandler)

    CUnboundHandler();          // protected constructor used by dynamic creation

DECLARE_INTERFACE_MAP()

public:
    BEGIN_INTERFACE_PART(Handler, EXLISTLib::IUnboundHandler)
            STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
        STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source);
    END_INTERFACE_PART(Handler)

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUnboundHandler)
    //}}AFX_VIRTUAL

// Implementation
    virtual ~CUnboundHandler();

    // Generated message map functions
    //{{AFX_MSG(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};
```

- Add INTERFACE_PART definition in the UnboundHandler.cpp file like follows:

```
BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
    INTERFACE_PART(CUnboundHandler, __uuidof(EXLISTLib::IUnboundHandler),
Handler)
```

```
END_INTERFACE_MAP()
```

- Write the get_ItemsCount property like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( pVal )
    {
        *pVal = 25000;
        return S_OK;;
    }
    return E_POINTER;
}
```

- Write the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);

    // gets the source control
    EXLISTLib::IList* pList = NULL;
    if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXLISTLib::IList),
(LPVOID*)&pList ) ) )
    {
        // assigns the value for each cell.
        pList->Items->Caption[ pList->Items->VirtualToItem[Index] ][
_variant_t((long)0) ] = _variant_t( Index );
        pList->Release();
    }
    return S_OK;
}
```

- Add implementation for QueryInterface, AddRef and Release methods of IUnknown interface like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   if ( ppvObject )
   {
      if ( IsEqualIID( __uuidof(IUnknown), riid ) )
      {
         *ppvObject = static_cast<IUnknown*>( this );
         AddRef();
         return S_OK;
      }
      if ( IsEqualIID( __uuidof( EXLISTLib::IUnboundHandler), riid ) )
      {
         *ppvObject = static_cast<EXLISTLib::IUnboundHandler*>( this );
         AddRef();
         return S_OK;
      }
      return E_NOINTERFACE;
   }
   return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   return 1;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   return 0;
}
```

- The CUnboundHandler class implementation should look like:

```
IMPLEMENT_DYNCREATE(CUnboundHandler, CCmdTarget)

BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
    INTERFACE_PART(CUnboundHandler, __uuidof(EXLISTLib::IUnboundHandler),
Handler)
END_INTERFACE_MAP()

CUnboundHandler::CUnboundHandler()
{
}

CUnboundHandler::~CUnboundHandler()
{
}

STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( pVal )
    {
        *pVal = 25000;
        return S_OK;;
    }
    return E_POINTER;
}

STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);

    // gets the source control
    EXLISTLib::IList* pList = NULL;
    if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXLISTLib::IList),
(LPVOID*)&pList ) ) )
    {
```

```
        // assigns the value for each cell.
        pList->Items->Caption[ pList->Items->VirtualToItem[Index] ][
_variant_t((long)0) ] = _variant_t( Index );
        pList->Release();
    }
    return S_OK;
}

STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( ppvObject )
    {
        if ( IsEqualIID( __uuidof(IUnknown), riid ) )
        {
            *ppvObject = static_cast<IUnknown*>( this );
            AddRef();
            return S_OK;
        }
        if ( IsEqualIID( __uuidof( EXLISTLib::IUnboundHandler), riid ) )
        {
            *ppvObject = static_cast<EXLISTLib::IUnboundHandler*>( this );
            AddRef();
            return S_OK;
        }
        return E_NOINTERFACE;
    }
    return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}
```

```
STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 0;
}

BEGIN_MESSAGE_MAP(CUnboundHandler, CCmdTarget)
    //{{AFX_MSG_MAP(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

After all these steps we have defined the class CUnboundHandler that implements the IUnboundHandler interface. All that we need to do from now, is to add a column to the control, and to set the VirtualMode and UnboundHanlder properties like follows:

- Open the definition of the application's main dialog (CADOVirtualDlg)
- Include the definition of the CUnboundHandler class to CADOVirtualDlg using:

```
#include "UnboundHandler.h"
```

- Add a new member of CUnboundHandler type to the CADOVirtualDlg class like:

```
CUnboundHandler m_unboundHandler;
```

- Open the implementation file for the application's main dialog (CADOVirtualDlg)
- Add the definition for CColumns class (a wrapper class for the control) at the beginning of the file

```
 #include "Columns.h"
```

- Locate the OnInitDialog() method and add the following code (after the "// TODO: Add extra initialization here"):

```
m_list.BeginUpdate();
    m_list.GetColumns().Add( _T("Column 1") );
    m_list.SetVirtualMode( TRUE );
    m_list.SetUnboundHandler( &m_unboundHandler.m_xHandler );
m_list.EndUpdate();
```

- Save, Compile and Run the project

The tutorial shows how to put the control on virtual mode. The sample loads the numbers from 0 to 24999.

Now, that we got the idea how to implement the IUnboundHandler let's say that we want to change the sample to load an edit an ADO recordset. The following tutorials shows how to display a table and how to add code in order to let user edits the data.

- Open the definition of the CUnboundHandler class
- Import the Microsoft ADO Type Library to the CUnboundHandler class like follows:

```
#import <msado15.dll> rename ( "EOF", "adoEOF" )
```

The #import directive generates the ADODB namspace. The ADODB namspace includes all definitions in the Microsoft ADO Type Library.

- Include a member of ADODB::_RecordsetPtr called m_spRecordset. The m_spRecordset member will handle data in the ADO table.

```
ADODB::_RecordsetPtr m_spRecordset;
```

- Add definition for AttachTable function like follows:

```
virtual void AttachTable( EXLISTLib::IList* pList, LPCTSTR szTable, LPCTSTR
szDatabase );
```

Now, the CUnboundHandler class definition should look like follows:

```
#import "c:\winnt\system32\exlist.dll"  rename( "GetItems", "exGetItems" )
#import <msado15.dll> rename ( "EOF", "adoEOF" )

class CUnboundHandler : public CCmdTarget
{
   DECLARE_DYNCREATE(CUnboundHandler)

   CUnboundHandler();          // protected constructor used by dynamic creation

DECLARE_INTERFACE_MAP()

public:
   BEGIN_INTERFACE_PART(Handler, EXLISTLib::IUnboundHandler)
            STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
```

```
    STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source);
  END_INTERFACE_PART(Handler)

  virtual void AttachTable( EXLISTLib::IList* pList, LPCTSTR szTable, LPCTSTR
szDatabase );

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CUnboundHandler)
  //}}AFX_VIRTUAL

// Implementation
  virtual ~CUnboundHandler();

  // Generated message map functions
  //{{AFX_MSG(CUnboundHandler)
    // NOTE - the ClassWizard will add and remove member functions here.
  //}}AFX_MSG

  DECLARE_MESSAGE_MAP()

  ADODB::_RecordsetPtr m_spRecordset;
};
```

- Open the implementation file for CUnboundHandler class (UnboundHandler.cpp file)
- Add the implementation for AttachTable function like follows:

```
void CUnboundHandler::AttachTable( EXLISTLib::IList* pList, LPCTSTR szTable,
LPCTSTR szDatabase )
{
  if ( SUCCEEDED( m_spRecordset.CreateInstance( "ADODB.Recordset") ) )
  {
    try
    {
      CString strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=";
      strConnection += szDatabase;
      if ( SUCCEEDED( m_spRecordset->Open(_variant_t( szTable ),
```

```
_variant_t(strConnection), ADODB::adOpenStatic, ADODB::adLockPessimistic,
NULL ) ) )
        {
            pList->BeginUpdate();
            for ( long i = 0; i < m_spRecordset->Fields->GetCount(); i++ )
                pList->GetColumns()->Add( m_spRecordset->Fields->GetItem(
_variant_t( i ) )->Name );
            pList->EndUpdate();
        }
    }
    catch ( _com_error& e )
    {
        AfxMessageBox( e.Description() );
    }

  }
}
```

The AttachTable function opens a recordset, and adds a new column to the control's Columns collection for each field found in the recordset.

- Change the get_ItemsCount property like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
  METHOD_PROLOGUE(CUnboundHandler, Handler);
  if ( pVal )
  {
    *pVal = pThis->m_spRecordset->RecordCount;
    return S_OK;;
  }
  return E_POINTER;
}
```

The ItemsCount property specifies that the control displays all records in the recordset

- Change the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

   // gets the source control
   EXLISTLib::IList* pList = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXLISTLib::IList),
(LPVOID*)&pList ) ) )
   {
      // assigns the value for each cell.
      long l = pList->Items->VirtualToItem[ Index ];
      for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
         pList->Items->Caption[ pList->Items->VirtualToItem[ Index ] ][ _variant_t(
i )] = pThis->m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;
      pList->Release();
   }
   return S_OK;
}
```

The ReadItem method moves the position of the current record in the recordset, and sets the value for each cell in the item.

The implementation for CUnbundHandler class should look like:

```
IMPLEMENT_DYNCREATE(CUnboundHandler, CCmdTarget)

BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
   INTERFACE_PART(CUnboundHandler, __uuidof(EXLISTLib::IUnboundHandler),
Handler)
END_INTERFACE_MAP()

CUnboundHandler::CUnboundHandler()
{
}
```

```
CUnboundHandler::~CUnboundHandler()
{
}

STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   if ( pVal )
   {
      *pVal = pThis->m_spRecordset->RecordCount;
      return S_OK;;
   }
   return E_POINTER;
}

STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

   // gets the source control
   EXLISTLib::IList* pList = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXLISTLib::IList),
(LPVOID*)&pList ) ) )
   {
      // assigns the value for each cell.
      long l = pList->Items->VirtualToItem[ Index ];
       for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
          pList->Items->Caption[ l ][ _variant_t( i )] = pThis-
>m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;
      pList->Release();
   }
   return S_OK;
}
```

```
void CUnboundHandler::AttachTable( EXLISTLib::IList* pList, LPCTSTR
szTable, LPCTSTR szDatabase )
{
    if ( SUCCEEDED( m_spRecordset.CreateInstance( "ADODB.Recordset") ) )
    {
        try
        {
            CString strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=";
            strConnection += szDatabase;
            if ( SUCCEEDED( m_spRecordset->Open(_variant_t( szTable ),
_variant_t(strConnection), ADODB::adOpenStatic,
ADODB::adLockPessimistic, NULL ) ) )
            {
                pList->BeginUpdate();
                for ( long i = 0; i < m_spRecordset->Fields->GetCount(); i++ )
                    pList->GetColumns()->Add( m_spRecordset->Fields-
>GetItem( _variant_t( i ) )->Name );
                pList->EndUpdate();
            }
        }
        catch ( _com_error& e )
        {
            AfxMessageBox( e.Description() );
        }
    }
}

STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( ppvObject )
    {
        if ( IsEqualIID( __uuidof(IUnknown), riid ) )
        {
```

```
            *ppvObject = static_cast<IUnknown*>( this );
            AddRef();
            return S_OK;
        }
        if ( IsEqualIID( __uuidof( EXLISTLib::IUnboundHandler), riid ) )
        {
            *ppvObject = static_cast<EXLISTLib::IUnboundHandler*>( this );
            AddRef();
            return S_OK;
        }
        return E_NOINTERFACE;
    }
    return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 0;
}

BEGIN_MESSAGE_MAP(CUnboundHandler, CCmdTarget)
    //{{AFX_MSG_MAP(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

- Locate the OnInitDialog method in the implementation file of the application's main dialog (AdoVirtualDlg.cpp)
- Add the following code to the OnInitDialog method:

```
EXLISTLib::IListPtr spList = NULL;
```

```
m_list.GetControlUnknown()->QueryInterface( &spList );
m_list.BeginUpdate();
    m_unboundHandler.AttachTable( spList, _T("Orders"),
_T("D:\\Exontrol\\ExList\\sample\\sample.mdb") );
    m_list.SetVirtualMode( TRUE );
    m_list.SetUnboundHandler( &m_unboundHandler.m_xHandler );
m_list.EndUpdate();
```

The AttachTable function is called before setting the UnboundHandler property. The AttachTable function opens a recordset giving the SQL phrase and the database. The AttachTable function loads also the control's Columns collection from the Fields collection of the recordset.

- Save, Compile and Run the project

After all these your control will be able to display a table using the virtual mode. Now, we need to add some changes in order to let user edits the data in the control.

- Change the AllowEdit property of the control like follows:

```
m_list.SetAllowEdit( TRUE );
```

The OnInitDialog looks like:

```
EXLISTLib::IListPtr spList = NULL;
m_list.GetControlUnknown()->QueryInterface( &spList );
m_list.BeginUpdate();
    m_unboundHandler.AttachTable( spList, _T("Orders"),
_T("D:\\Exontrol\\ExList\\sample\\sample.mdb") );
    m_list.SetAllowEdit( TRUE );
    m_list.SetVirtualMode( TRUE );
    m_list.SetUnboundHandler( &m_unboundHandler.m_xHandler );
m_list.EndUpdate();
```

- Add the definition for the CItems class to CAdoVirtualDlg implementation file:

```
#include "Items.h"
```

- Add a new handler for AfterCellEdit event:

```
void CADOVirtualDlg::OnAfterCellEditList1(long Index, long ColIndex, LPCTSTR
```

```
NewCaption)
{
    m_unboundHandler.Change( NewCaption, m_list.GetItems().GetItemToVirtual(
Index ), ColIndex );
}
```

- Add a new function definition (Change) to the CUnboundHandler class:

```
virtual void Change( LPCTSTR szCaption, long Index, long ColIndex );
```

The CUnboundHandler class definition should look like:

```
#import "c:\winnt\system32\exlist.dll"  rename( "GetItems", "exGetItems" )
#import <msado15.dll> rename ( "EOF", "adoEOF" )

class CUnboundHandler : public CCmdTarget
{
    DECLARE_DYNCREATE(CUnboundHandler)

    CUnboundHandler();          // protected constructor used by dynamic creation

DECLARE_INTERFACE_MAP()

public:
    BEGIN_INTERFACE_PART(Handler, EXLISTLib::IUnboundHandler)
            STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
        STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source);
    END_INTERFACE_PART(Handler)

    virtual void AttachTable( EXLISTLib::IList* pList, LPCTSTR szTable, LPCTSTR
szDatabase );
    virtual void Change( LPCTSTR szCaption, long Index, long ColIndex );

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUnboundHandler)
    //}}AFX_VIRTUAL
```

```
// Implementation
    virtual ~CUnboundHandler();

    // Generated message map functions
    //{{AFX_MSG(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()

    ADODB::_RecordsetPtr m_spRecordset;
};
```

- Add the Change function's body like follows:

```
void CUnboundHandler::Change( LPCTSTR szCaption, long Index, long ColIndex )
{
    m_spRecordset->Move( Index, _variant_t( (long)ADODB::adBookmarkFirst ) );
    m_spRecordset->Fields->GetItem( _variant_t( ColIndex ) )->Value = szCaption;
    m_spRecordset->Update();
}
```

- Save, Compile and Run the project.

If you need to apply colors, font attributes, … for items or cells while the control is running in the virtual mode, the changes should be done in the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

    // gets the source control
    EXLISTLib::IList* pList = NULL;
    if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXLISTLib::IList),
(LPVOID*)&pList ) ) )
    {
```

```
        long l = pList->Items->VirtualToItem[ Index ];
        // assigns the value for each cell.
        for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
            pList->Items->Caption[ l ][ _variant_t( i )] = pThis->m_spRecordset-
>Fields->GetItem( _variant_t( i ) )->Value;

        if ( pList->Items->Caption[ _variant_t( l ) ][ _variant_t( _T("ShipRegion") )] ==
_variant_t( _T("RJ") ) )
            pList->Items->put_ItemForeColor( l , RGB(0,0,255 ) );
        if ( pList->Items->Caption[ _variant_t( l ) ][ _variant_t( _T("ShipRegion") )] ==
_variant_t( _T("SP") ) )
            pList->Items->put_ItemBold( l , TRUE );

        pList->Release();
    }
    return S_OK;
}
```

While compiling the project the compiler displays warnings like: "warning C4146: unary minus operator applied to unsigned type, result still unsigned". You have to include the :

```
#pragma warning( disable : 4146 )
```

before importing the type libraries.

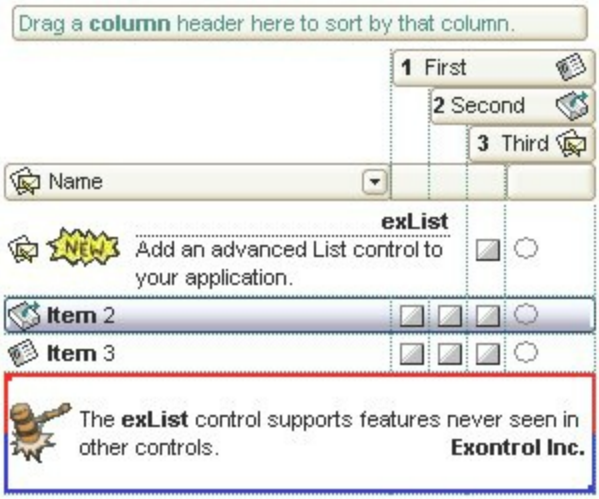
```
#pragma warning( disable : 4146 )
#import "c:\winnt\system32\exlist.dll" rename( "GetItems", "exGetItems" )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
```

property List.VisualAppearance as Appearance

Retrieves the control's appearance.

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

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



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

- control's **header bar**, BackColorHeader property
- control's **filter bar**, FilterBarBackColor property
- control's **sort bar**, BackColorSort property
- the caption of the control's sort bar, BackColorSortCaption property
- **selected item** or cell, SelBackColor property
- **item**, ItemBackColor property
- **cell**, CellBackColor property
- cell's **button**, "**drop down**" filter bar button, "close" filter bar button, and so on, Background property

property List.VisualDesign as String

Invokes the control's VisualAppearance designer.

Type	Description
String	A String expression that encodes the control's Visual Appearance.

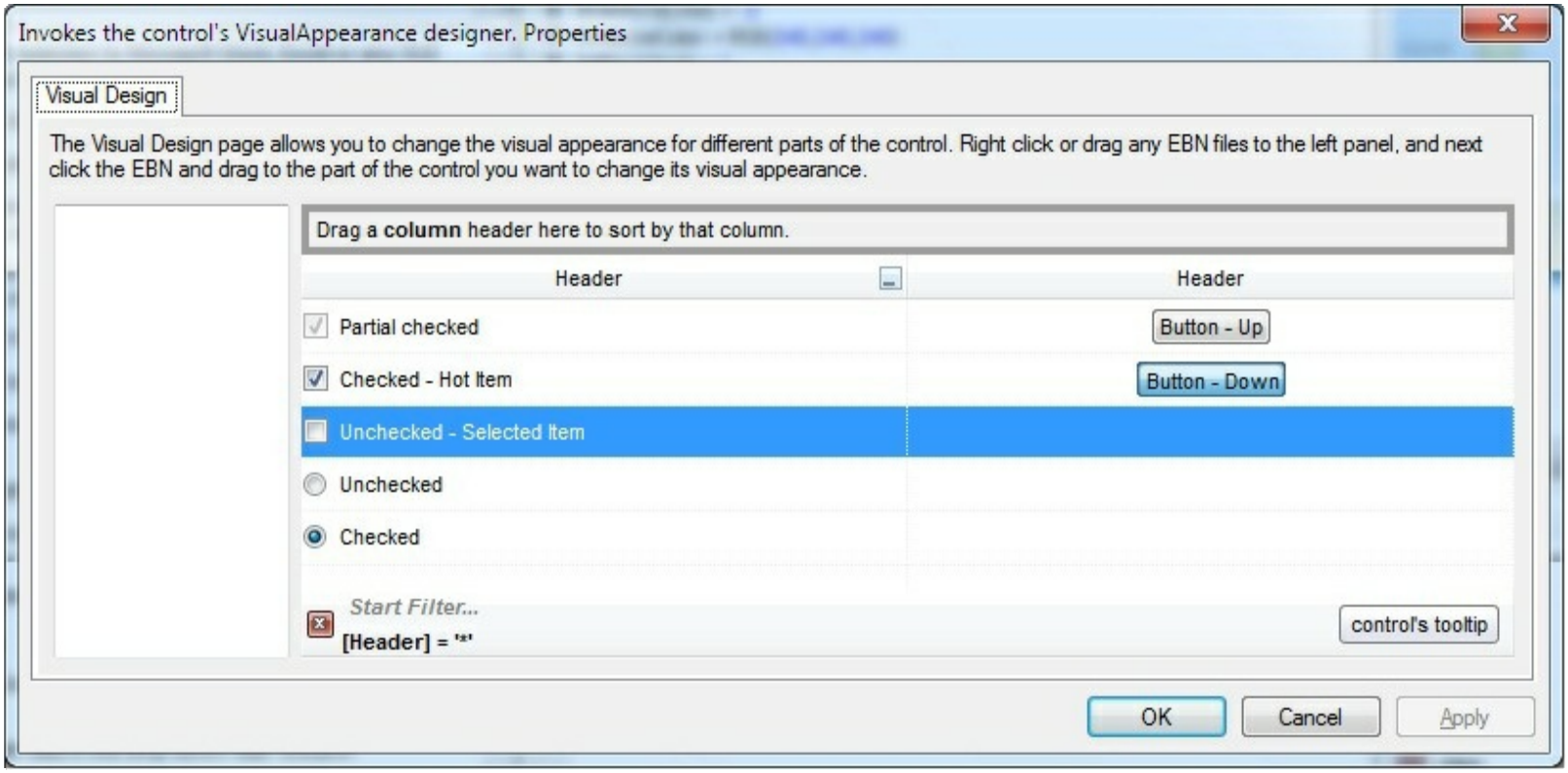
By default, the VisualDesign property is "". The VisualDesign property helps you to define fast and easy the control's visual appearance using the XP-Theme elements or EBN objects. The VisualDesign property can be accessed on design mode, and it can be used to design the visual appearance of different 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.

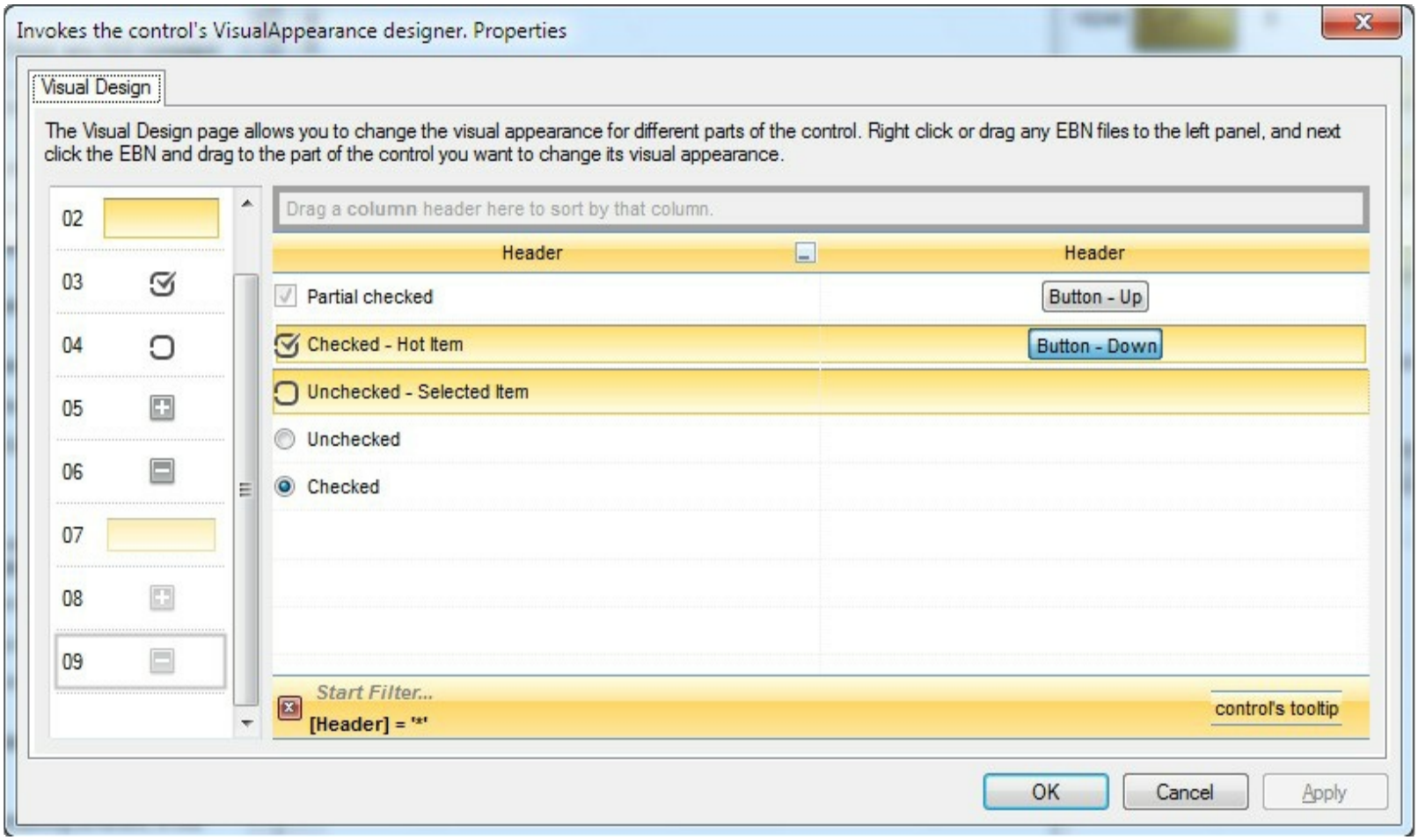
- Click here to watch a movie on how you define the control's visual appearance using the XP-Theme
- Click here to watch a movie on how you define the control's visual appearance using the EBN files.

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 List1
   .VisualDesign =
"gBFLBWIgBAEHhEJAEGg7oB0HBSQAwABsIfj/jEJAcKhYEjgCAscA8ThQBA8cAgIjgDh8KBAPj
 & _

"RuF6FxmAkchiheZg5gYZIW0yMhZhqD55jIboamcCY2HGG5nCmVh0h2ZYUAyCQ4Xqbh9h8
 & _

"o5B8MwE4HsD4/g/ijHQHoLwrxUjrH0H4Z4rR2h7A8N8UggRNBnGCP8eA/A/gXGSPMfg3w
 & _

"DCDgJQFICxhDQGYBofYQYFCwD4J+XYQwIBECiCwJIExhhnCIDoNAnhzj8CyBcIosQ+BlAwM
 & _

"J8YQlwaBMCaCMd6hRnBpE+HoIwIQ9hdEKM8VYawoCcC8BUSYtxqBuDuFsOwTgLgLhZhAh
 & _

"zhGhtoEB+AsArhnhLhehUB5BfA4BfARBPgWB9h3hhBZB/AvA+BzhkhLhCh7hPg8g1BfhzAKE
 & _

"hQH1hSgAgcAmghgIg2AugLBigiBqAnAzBiVdgIA1ANAjBEgbAmAJMwA+gLgjgyBWA4A0E
 & _

"IAUgCA0AMhjA0ggWUgjh+GhBihI1yAKhiByBqAkV1gCAKAiV3141516g+Jmhj19V+V/AI2A

End With
```

If running the empty control we get the following picture:

A	B	=(A+B)*1.19
1	110	132.09
2	22	28.56
2	99	120.19
1	89	107.1
3	11	16.66
1	6	8.33

If running the control using the code being generated by the VisualAppearance designer we get:

A	B	=(A+B)*1.19
1	110	**132.09**
2	22	**28.56**
2	99	**120.19**
1	89	**107.1**
3	11	**16.66**
1	6	**8.33**

UnboundHandler object

The control supports unbound mode. In unbound mode, the user is responsible for retrieving items. The unbound mode and virtual unbound modes were provided to let user displays large number of items. In order to let the control works in unbound mode, the user has to implement the IUnboundHandler notification interface. The UnboundHandler property specifies the control's unbound handler. Currently, the UnboundHandler / IUnboundHandler interface is available for /COM version only. Use the VirtualMode property to run the control in virtual mode. Use the VirtualToItem property to get the index of the item in the list giving the index of the virtual item.

Here's the IDL definition of the IUnboundHandler interface:

```
[
   uuid(42FD4C01-3385-4BF4-9F42-E6E65104CF42),
   pointer_default(unique)
]
interface IUnboundHandler : IUnknown
{
   [propget, id(1), helpcontext(3001), helpstring("Gets the number of items.")] HRESULT
ItemsCount( IDispatch* Source, [out, retval ] long* pVal );
   [id(2), helpcontext(3002), helpstring("The source requires an item.")] HRESULT ReadItem(
long Index, IDispatch* Source );
}
```

Here's the IDL definition of the UnboundHandler interface (this interface is available starting from the version 11.1):

```
[
   uuid(42FD4C01-3385-4BF4-9F42-E6E65104CF43),
]
dispinterface UnboundHandler
{
   interface IUnboundHandler;
}
```

The following **VB** sample displays 1,000,000 items in virtual mode:

```
Implements EXLISTLibCtl.IUnboundHandler
```

```
Private Sub Form_Load()
   With List1
      .BeginUpdate
      .Columns.Add("Index").FormatColumn = "value format `0`"
      .VirtualMode = True
      Set .UnboundHandler = Me
      .EndUpdate
   End With
End Sub

Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = 1000000
End Property

Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object)
   With Source.Items
      .Caption(.VirtualToItem(Index), 0) = Index + 1
   End With
End Sub
```

The following **VB/NET** sample displays 1,000,000 items in virtual mode:

```
Public Class Form1
   Implements EXLISTLib.IUnboundHandler

   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
      With AxList1
         .BeginUpdate()
         .Columns.Add("Index").FormatColumn = "value format `0`"
         .VirtualMode = True
         .UnboundHandler = Me
         .EndUpdate()
      End With
   End Sub

   Public ReadOnly Property ItemsCount(ByVal Source As Object) As Integer Implements
```

```
EXLISTLib.IUnboundHandler.ItemsCount
        Get
            ItemsCount = 10000000
        End Get
    End Property

    Public Sub ReadItem(ByVal Index As Integer, ByVal Source As Object) Implements
EXLISTLib.IUnboundHandler.ReadItem
        With Source.Items
            .Caption(.VirtualToItem(Index), 0) = Index + 1
        End With
    End Sub
End Class
```

The following **C#** sample displays 1,000,000 items in virtual mode:

```
public partial class Form1 : Form, EXLISTLib.IUnboundHandler
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        axList1.BeginUpdate();
        (axList1.Columns.Add("Index") as EXLISTLib.IColumn).FormatColumn = "value format
`0`";
        axList1.VirtualMode = true;
        axList1.UnboundHandler = this;
        axList1.EndUpdate();
    }

    public int get_ItemsCount(object Source)
    {
        return 1000000;
    }
```

```
    public void ReadItem(int Index, object Source)
    {
        (Source as EXLISTLib.IList).Items.set_Caption((Source as
EXLISTLib.IList).Items.get_VirtualToItem(Index), 0, Index + 1);
    }
}
```

The following **VFP 9** sample displays 1,000,000 items in virtual mode:

```
with thisform.List1
    .Columns.Add("Index").FormatColumn = "value format `0`"
    .VirtualMode = .T.
    .UnboundHandler = newobject('UnboundHandler', 'class1.prg')
endwith
```

where the class1.prg is:

```
define class UnboundHandler as session OLEPUBLIC

implements IUnboundHandler in "ExList.dll"
function IUnboundHandler_get_ItemsCount(Source)
    return 1000000
endfunc
function IUnboundHandler_ReadItem(Index, Source)
    With Source.Items
        .Caption(.VirtualToItem(Index), 0) = Index + 1
    EndWith
endfunc

implements UnboundHandler in "ExList.dll"
function UnboundHandler_get_ItemsCount(Source)
    return this.IUnboundHandler_get_ItemsCount(Source)
endfunc
function UnboundHandler_ReadItem(Index, Source)
    return this.IUnboundHandler_ReadItem(Index, Source)
endfunc
```

```
enddefine
```

The following **VFP 7** and **VFP 8** sample displays 1,000,000 items in virtual mode:

```
with thisform.List1
   .Columns.Add("Index").FormatColumn = "value format `0`"
   .VirtualMode = .T.
   .UnboundHandler = newobject('UnboundHandler', 'class1.prg')
endwith
```

where the class1.prg is:

```
define class UnboundHandler as custom

implements IUnboundHandler in "ExList.dll"
function IUnboundHandler_get_ItemsCount(Source)
   return 1000000
endfunc
function IUnboundHandler_ReadItem(Index, Source)
   With Source.Items
      .Caption(.VirtualToItem(Index), 0) = Index + 1
   EndWith
endfunc

enddefine
```

The UnboundHandler / IUnboundHandler interface requires the following properties and methods:

Name	Description
ItemsCount	Gets the number of items.
ReadItem	The source requires an item.

property UnboundHandler.ItemsCount (Source as Object) as Long

Gets the number of items.

Type	Description
Source as Object	The control that requires the number of items
Long	A Long expression that specifies the number of items in unbound/virtual mode.

The ItemsCount property specifies the number of items in unbound/virtual mode.

The following **VB** sample, shows how ItemsCount property should be implemented:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = 1000000
End Property
```

The following **VB/NET** sample, shows how ItemsCount property should be implemented:

```
Public ReadOnly Property ItemsCount(ByVal Source As Object) As Integer Implements
EXLISTLib.IUnboundHandler.ItemsCount
   Get
      ItemsCount = 10000000
   End Get
End Property
```

The following **C#** sample, shows how ItemsCount property should be implemented:

```
public int get_ItemsCount(object Source)
{
   return 1000000;
}
```

The following **VFP** sample , shows how ItemsCount property should be implemented:

```
function IUnboundHandler_get_ItemsCount(Source)
   return 1000000
endfunc
```

method UnboundHandler.ReadItem (Index as Long, Source as Object)

The source requires an item.

Type	Description
Index as Long	A Long expression that specifies the index of the item to be requested. Use the VirtualToItem property to get the index of the item in the list giving the index of the virtual item.
Source as Object	The source object to be filled.

The ReadItem method is called every time the Source requires an item to be displayed.

The following **VB** sample, shows how ReadItem method should be implemented:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object)
   With Source.Items
      .Caption(.VirtualToItem(Index), 0) = Index + 1
   End With
End Sub
```

The following **VB/NET** sample, shows how ReadItem method should be implemented:

```
Public Sub ReadItem(ByVal Index As Integer, ByVal Source As Object) Implements
EXLISTLib.IUnboundHandler.ReadItem
   With Source.Items
      .Caption(.VirtualToItem(Index), 0) = Index + 1
   End With
End Sub
```

The following **C#** sample, shows how ReadItem method should be implemented:

```
public void ReadItem(int Index, object Source)
{
   (Source as EXLISTLib.IList).Items.set_Caption((Source as
EXLISTLib.IList).Items.get_VirtualToItem(Index), 0, Index + 1);
}
```

The following **VFP** sample , shows how ReadItem method should be implemented:

```
function IUnboundHandler_ReadItem(Index, Source)
```

```
    With Source.Items
        .Caption(.VirtualToItem(Index), 0) = Index + 1
    EndWith
endfunc
```

ExList events

The ExList control supports the following events:

Name	Description
AddColumn	Fired after a new column is added.
AddItem	Occurs after a new Item is inserted to Items collection.
AfterCellEdit	Occurs after data in the current cell is edited.
AllowAutoDrag	Occurs when the user drags the item between InsertA and InsertB.
AnchorClick	Occurs when an anchor element is clicked.
BeforeCellEdit	Occurs just before the user enters edit mode by clicking in a cell.
CancelCellEdit	Occurs if the edit operation is canceled.
CellButtonClick	Fired after the user clicks the cell's button.
CellImageClick	Fired after the user clicks the cell's icon.
CellStateChanged	Fired after cell's state is changed.
CellStateChanging	Fired before cell's state is about to be changed.
Click	Occurs when the user presses and then releases the left mouse button over the list control.
ColumnClick	Fired after the user clicks on column's header.
DblClick	Occurs when the user presses and releases a mouse button and then presses and releases it again over an object.
Event	Notifies the application once the control fires an event.
FilterChange	Occurs when filter was changed.
FilterChanging	Notifies your application that the filter is about to change.
FormatColumn	Fired when a cell requires to format its caption.
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.
LayoutChanged	Occurs when column's position or column's size is changed.

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.
OffsetChanged	Occurs when the scroll position is changed.
OLECompleteDrag	Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled
OLEDragDrop	Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.
OLEDragOver	Occurs when one component is dragged over another.
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.
OversizeChanged	Occurs when the right range of the scroll is changed.
RClick	Fired when right mouse button is clicked
RemoveColumn	Fired before deleting a Column.
RemoveItem	Occurs before deleting an Item.
ScrollButtonClick	Occurs when the user clicks a button in the scrollbar.
SelectionChanged	Fired after a new item is selected.
Sort	Fired when the control sorts a column.
ToolTip	Fired when the control prepares the object's tooltip.

event AddColumn (Column as Column)

Fired after a new column has been added.

Type	Description
Column as Column	A Column object that's added to the Columns collection.

The control fires the AddColumn event is fired when a new column is inserted to the Columns collection. Use the AddColumn event to associate any extra data to the column. Use the Add method to add a new column. Use the Add, PutItems or DataSource method to add new items to the control. Use the ColumnAutoResize property to allow visible columns fit the control's client area.

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

C#

```
private void AddColumn(object sender,exontrol.EXLISTLib.Column Column)
{
}
```

VB

```
Private Sub AddColumn(ByVal sender As System.Object,ByVal Column As
exontrol.EXLISTLib.Column) Handles AddColumn
End Sub
```

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

C#

```
private void AddColumn(object sender,
AxEXLISTLib._IListEvents_AddColumnEvent e)
{
}
```

C++

```
void OnAddColumn(LPDISPATCH Column)
{
}
```

C++ Builder

```
void __fastcall AddColumn(TObject *Sender,Exlistlib_tlb::IColumn *Column)
{
}
```

Delphi

```
procedure AddColumn(ASender: TObject; Column : IColumn);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure AddColumn(sender: System.Object; e:
AxEXLISTLib._IListEvents_AddColumnEvent);
begin
end;
```

Powe…

```
begin event AddColumn(oleobject Column)
end event AddColumn
```

VB.NET

```
Private Sub AddColumn(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_AddColumnEvent) Handles AddColumn
End Sub
```

VB6

```
Private Sub AddColumn(ByVal Column As EXLISTLibCtl.IColumn)
End Sub
```

VBA

```
Private Sub AddColumn(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnAddColumn(oList,Column)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComAddColumn Variant llColumn
    Forward Send OnComAddColumn llColumn
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_AddColumn(Column) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddColumn(COM _Column)
{
}
```

XBasic

```
function AddColumn as v (Column as OLE::Exontrol.List.1::IColumn)
end function
```

dBASE

```
function nativeObject_AddColumn(Column)
return
```

The following VB sample sets the width for all columns:

```
Private Sub List1_AddColumn(ByVal Column As EXLISTLibCtl.IColumn)
   Column.Width = 128
End Sub
```

The following C++ sample changes the column's width when a new column is added:

```
#include "Column.h"
#include "Columns.h"
void OnAddColumnList1(LPDISPATCH Column)
{
   CColumn column( Column );column.m_bAutoRelease = FALSE;
   column.SetWidth( 128 );
}
```

The following VB.NET sample changes the column's width when a new column is added:

```
Private Sub AxList1_AddColumn(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_AddColumnEvent) Handles AxList1.AddColumn
   e.column.Width = 128
End Sub
```

The following C# sample changes the column's width when a new column is added:

```
private void axList1_AddColumn(object sender, AxEXLISTLib._IListEvents_AddColumnEvent
e)
{
    e.column.Width = 128;
}
```

The following VFP sample changes the column's width when a new column is added:

```
*** ActiveX Control Event ***
LPARAMETERS column

with column
    .Width = 128
endwith
```

event AddItem (ItemIndex as Long)

Occurs after a new item has been inserted to Items collection.

Type	Description
ItemIndex as Long	A long expression that indicates the index of the item being added.

The control fires the AddItem event when a new item is added. Use the Add method to add new items to the control. Use the AddItem event to associate extra data to the item. Use the Add method to add a new column to the control. Use the PutItems to add items from safe array. Use the DataSource property to add columns and items from a recordset.

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

C#

```
private void AddItem(object sender,int ItemIndex)
{
}
```

VB

```
Private Sub AddItem(ByVal sender As System.Object,ByVal ItemIndex As Integer)
Handles AddItem
End Sub
```

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

C#

```
private void AddItem(object sender, AxEXLISTLib._IListEvents_AddItemEvent e)
{
}
```

C++

```
void OnAddItem(long ItemIndex)
{
}
```

C++ Builder

```
void __fastcall AddItem(TObject *Sender,long ItemIndex)
{
}
```

Delphi

```
procedure AddItem(ASender: TObject; ItemIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AddItem(sender: System.Object; e:
AxEXLISTLib._IListEvents_AddItemEvent);
begin
end;
```

Powe…

```
begin event AddItem(long ItemIndex)
end event AddItem
```

VB.NET

```
Private Sub AddItem(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_AddItemEvent) Handles AddItem
End Sub
```

VB6

```
Private Sub AddItem(ByVal ItemIndex As Long)
End Sub
```

VBA

```
Private Sub AddItem(ByVal ItemIndex As Long)
End Sub
```

VFP

```
LPARAMETERS ItemIndex
```

Xbas…

```
PROCEDURE OnAddItem(oList,ItemIndex)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComAddItem Integer llItemIndex
    Forward Send OnComAddItem llItemIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_AddItem(ItemIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddItem(int _ItemIndex)
{
}
```

XBasic

```
function AddItem as v (ItemIndex as N)
end function
```

dBASE

```
function nativeObject_AddItem(ItemIndex)
return
```

The following VB sample bolds the cells in the first column, when a new items is inserted:

```
Private Sub List1_AddItem(ByVal Index As Long)
   With List1.Items
      .CellBold(Index, 0) = True
   End With
End Sub
```

The following C++ sample bolds the cells in the first column, when a new items is inserted:

```
void OnAddItemList1(long ItemIndex)
{
   if ( IsWindow( m_list.m_hWnd ) )
   {
      CItems items = m_list.GetItems();
      items.SetCellBold( ItemIndex, COleVariant( long( 0 ) ), TRUE );
   }
}
```

The following VB.NET sample bolds the cells in the first column, when a new items is inserted:

```
Private Sub AxList1_AddItem(ByVal sender As System.Object, ByVal e As
```

```
AxEXLISTLib._IListEvents_AddItemEvent) Handles AxList1.AddItem
    With AxList1.Items
        .CellBold(e.itemIndex, 0) = True
    End With
End Sub
```

The following C# sample bolds the cells in the first column, when a new items is inserted:

```
private void axList1_AddItem(object sender, AxEXLISTLib._IListEvents_AddItemEvent e)
{
    axList1.Items.set_CellBold(e.itemIndex, 0, true);
}
```

The following VFP sample bolds the cells in the first column, when a new items is inserted:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex

with thisform.List1.Items
    .CellBold( itemIndex, 0) = .t.
endwith
```

event AfterCellEdit (ItemIndex as Long, ColIndex as Long, NewCaption as String)

Occurs after data in the current cell is edited.

Type	Description
ItemIndex as Long	A long expression that indicates the index of item being edited.
ColIndex as Long	A long expression that indicates the column's index
NewCaption as String	A string expression that indicates the text being edited

Use the AfterCellEdit event to change the cell's caption after the edit operation ends. The control fires BeforeCellEdit event and AfterCellEdit events if the AllowEdit property is True. The AfterCellEdit event notifies your application that the user alters the cell's caption. Use the Edit method to edit programmatically a cell. The CancelCellEdit event occurs if the user cancels the edit operation. The CancelCellEdit event is fired when the user presses ESC key while editing or when the user clicks outside the edit field.

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

C#

```
private void AfterCellEdit(object sender,int ItemIndex,int ColIndex,string NewCaption)
{
}
```

VB

```
Private Sub AfterCellEdit(ByVal sender As System.Object,ByVal ItemIndex As Integer,ByVal ColIndex As Integer,ByVal NewCaption As String) Handles AfterCellEdit
End Sub
```

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

C#

```
private void AfterCellEdit(object sender, AxEXLISTLib._IListEvents_AfterCellEditEvent e)
{
}
```

C++

```
void OnAfterCellEdit(long ItemIndex,long ColIndex,LPCTSTR NewCaption)
{
}
```

C++ Builder

```
void __fastcall AfterCellEdit(TObject *Sender,long ItemIndex,long ColIndex,BSTR NewCaption)
{
}
```

Delphi

```
procedure AfterCellEdit(ASender: TObject; ItemIndex : Integer;ColIndex : Integer;NewCaption : WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterCellEdit(sender: System.Object; e: AxEXLISTLib._IListEvents_AfterCellEditEvent);
begin
end;
```

Powe…

```
begin event AfterCellEdit(long ItemIndex,long ColIndex,string NewCaption)
end event AfterCellEdit
```

VB.NET

```
Private Sub AfterCellEdit(ByVal sender As System.Object, ByVal e As AxEXLISTLib._IListEvents_AfterCellEditEvent) Handles AfterCellEdit
End Sub
```

VB6

```
Private Sub AfterCellEdit(ByVal ItemIndex As Long,ByVal ColIndex As Long,ByVal NewCaption As String)
End Sub
```

VBA

```
Private Sub AfterCellEdit(ByVal ItemIndex As Long,ByVal ColIndex As Long,ByVal NewCaption As String)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex,NewCaption
```

Xbas…

```
PROCEDURE OnAfterCellEdit(oList,ItemIndex,ColIndex,NewCaption)
RETURN
```

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

Java…

```
<SCRIPT EVENT="AfterCellEdit(ItemIndex,ColIndex,NewCaption)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AfterCellEdit(ItemIndex,ColIndex,NewCaption)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAfterCellEdit Integer llItemIndex Integer llColIndex String
llNewCaption
    Forward Send OnComAfterCellEdit llItemIndex llColIndex llNewCaption
End_Procedure
```

Visual Objects

```
METHOD OCX_AfterCellEdit(ItemIndex,ColIndex,NewCaption) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AfterCellEdit(int _ItemIndex,int _ColIndex,str _NewCaption)
{
}
```

XBasic

```
function AfterCellEdit as v (ItemIndex as N,ColIndex as N,NewCaption as C)
end function
```

dBASE

```
function nativeObject_AfterCellEdit(ItemIndex,ColIndex,NewCaption)
return
```

The following VB sample change the cell's caption when the user hits the ENTER key:

```
Private Sub List1_AfterCellEdit(ByVal Index As Long, ByVal ColIndex As Long, ByVal
NewCaption As String)
    Debug.Print NewCaption
    With List1.Items
        .Caption(Index, ColIndex) = NewCaption
    End With
End Sub
```

The following C++ sample change the cell's caption when the user hits the ENTER key:

```
void OnAfterCellEditList1(long ItemIndex, long ColIndex, LPCTSTR NewCaption)
{
   CItems items = m_list.GetItems();
   items.SetCaption( ItemIndex, COleVariant( ColIndex ), COleVariant( NewCaption ) );
}
```

The following VB.NET sample change the cell's caption when the user hits the ENTER key:

```
Private Sub AxList1_AfterCellEdit(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_AfterCellEditEvent) Handles AxList1.AfterCellEdit
   With AxList1.Items
      .Caption(e.itemIndex, e.colIndex) = e.newCaption
   End With
End Sub
```

The following C# sample change the cell's caption when the user hits the ENTER key:

```
private void axList1_AfterCellEdit(object sender,
AxEXLISTLib._IListEvents_AfterCellEditEvent e)
{
   axList1.Items.set_Caption(e.itemIndex, e.colIndex, e.newCaption);
}
```

The following VFP sample change the cell's caption when the user hits the ENTER key:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex, newcaption

with thisform.List1.Items
   .Caption( itemIndex, colindex) = newcaption
endwith
```

event AllowAutoDrag (Item as Long, InsertA as Long, InsertB as Long, Cancel as Boolean)

Occurs when the user drags the item between InsertA and InsertB.

Type	Description
Item as Long	A long expression that indicates the item being dragged.
InsertA as Long	A long expression that specifies the index of the item to insert the dragging Item after. If -1, it indicates that no item after.
InsertB as Long	A long expression that specifies the index of the item to insert the dragging Item before. If -1, it indicates that no item before.
Cancel as Boolean	A Boolean expression that specifies whether the operation can continue (this parameter is by reference)

The AllowAutoDrag event occurs when the user drags the item between InsertA and InsertB, using the AutoDrag property. The AutoDrag feature indicates what the control does when the user clicks an item and starts dragging it. For instance, using the AutoDrag feature you can let the user arrange the items in the control, or can drop the selection to a any OLE compliant applications like Microsoft Word, Excel and so on... The AllowAutoDrag event may fire when the AutoDrag property is any of the following values:

- exAutoDragPosition... (the item can be dragged from a position to another)

You can use the AllowAutoDrag event to cancel or continue drag and drop operation using the AutoDrag property.

The following screen shot shows the InsertA and InsertB parameters, when "Item 2" is dragging to a new position:

- InsertA is "Item 1"
- InsertB is "Item 3"

- InsertA is -1
- InsertB is "Item 1"

- InsertA is "Item 3"
- InsertB is -1

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

C#

```
private void AllowAutoDrag(object sender,int   Item,int   InsertA,int   InsertB,ref
bool   Cancel)
{
}
```

VB

```
Private Sub AllowAutoDrag(ByVal sender As System.Object,ByVal Item As
Integer,ByVal InsertA As Integer,ByVal InsertB As Integer,ByRef Cancel As Boolean)
Handles AllowAutoDrag
End Sub
```

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

C#

```
private void AllowAutoDrag(object sender,
AxEXLISTLib._IListEvents_AllowAutoDragEvent e)
{
}
```

C++

```
void OnAllowAutoDrag(long   Item,long   InsertA,long   InsertB,BOOL FAR*
Cancel)
{
}
```

C++ Builder

```
void __fastcall AllowAutoDrag(TObject *Sender,long   Item,long   InsertA,long
InsertB,VARIANT_BOOL *   Cancel)
{
}
```

Delphi

```
procedure AllowAutoDrag(ASender: TObject; Item : Integer;InsertA :
Integer;InsertB : Integer;var Cancel : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AllowAutoDrag(sender: System.Object; e:
AxEXLISTLib._IListEvents_AllowAutoDragEvent);
begin
end;
```

Powe…

```
begin event AllowAutoDrag(long  Item,long  InsertA,long  InsertB,boolean
Cancel)

end event AllowAutoDrag
```

VB.NET

```
Private Sub AllowAutoDrag(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_AllowAutoDragEvent) Handles AllowAutoDrag
End Sub
```

VB6

```
Private Sub AllowAutoDrag(ByVal Item As Long,ByVal InsertA As Long,ByVal
InsertB As Long,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub AllowAutoDrag(ByVal Item As Long,ByVal InsertA As Long,ByVal
InsertB As Long,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS Item,InsertA,InsertB,Cancel
```

Xbas…

```
PROCEDURE OnAllowAutoDrag(oList,Item,InsertA,InsertB,Cancel)
```

```
RETURN
```

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

Java…

```
<SCRIPT EVENT="AllowAutoDrag(Item,InsertA,InsertB,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AllowAutoDrag(Item,InsertA,InsertB,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAllowAutoDrag Integer  llItem Integer  llInsertA Integer
llInsertB Boolean  llCancel
   Forward Send OnComAllowAutoDrag llItem llInsertA llInsertB llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_AllowAutoDrag(Item,InsertA,InsertB,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AllowAutoDrag(int  _Item,int  _InsertA,int  _InsertB,COMVariant
/*bool*/  _Cancel)
{
}
```

XBasic

```
function AllowAutoDrag as v (Item  as  N,InsertA  as  N,InsertB  as  N,Cancel  as  L)
end function
```

dBASE

```
function nativeObject_AllowAutoDrag(Item,InsertA,InsertB,Cancel)
return
```

event AnchorClick (AnchorID as String, Options as String)

Occurs when an anchor element is clicked.

Type	Description
AnchorID as String	A string expression that indicates the identifier of the anchor
Options as String	A string expression that specifies options of the anchor element.

The control fires the AnchorClick event to notify that the user clicks an anchor element. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The **<a>** element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The AnchorClick event is fired only if prior clicking the control it shows the hand cursor. For instance, if the cell is disabled, the hand cursor is not shown when hovers the anchor element, and so the AnchorClick event is not fired. Use the FormatAnchor property to specify the visual effect for anchor elements. For instance, if the user clicks the anchor *<a1>anchor*, 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".

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, AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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 BeforeCellEdit (ItemIndex as Long, ColIndex as Long, Value as Variant, Cancel as Variant)

Occurs just before the user enters edit mode by clicking in a cell.

Type	Description
ItemIndex as Long	A long expression that indicates the index of item that being edited.
ColIndex as Long	A long expression that indicates the index of column being edited.
Value as Variant	A string value that indicates the cell's caption being edited.
Cancel as Variant	A boolean expression that indicates whether the edit operation is canceled or not.

The BeforeCellEdit event notifies your application that a cell starts editing. Use the BeforeCellEdit event to cancel editing a cell. The control fires the BeforeCellEdit and AfterCellEdit events only if the AllowEdit property is True. Use the Edit method to edit programmatically a cell.

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

C#
```
private void BeforeCellEdit(object sender,int ItemIndex,int ColIndex,ref object
Value,ref object Cancel)
{
}
```

VB
```
Private Sub BeforeCellEdit(ByVal sender As System.Object,ByVal ItemIndex As
Integer,ByVal ColIndex As Integer,ByRef Value As Object,ByRef Cancel As Object)
Handles BeforeCellEdit
End Sub
```

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

C#
```
private void BeforeCellEdit(object sender,
AxEXLISTLib._IListEvents_BeforeCellEditEvent e)
{
}
```

C++
```
void OnBeforeCellEdit(long ItemIndex,long ColIndex,VARIANT FAR*
Value,VARIANT FAR* Cancel)
```

```
{
}
```

C++ Builder

```
void __fastcall BeforeCellEdit(TObject *Sender,long ItemIndex,long ColIndex,Variant * Value,Variant * Cancel)
{
}
```

Delphi

```
procedure BeforeCellEdit(ASender: TObject; ItemIndex : Integer;ColIndex : Integer;var Value : OleVariant;var Cancel : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeCellEdit(sender: System.Object; e: AxEXLISTLib._IListEvents_BeforeCellEditEvent);
begin
end;
```

Powe…

```
begin event BeforeCellEdit(long ItemIndex,long ColIndex,any Value,any Cancel)
end event BeforeCellEdit
```

VB.NET

```
Private Sub BeforeCellEdit(ByVal sender As System.Object, ByVal e As AxEXLISTLib._IListEvents_BeforeCellEditEvent) Handles BeforeCellEdit
End Sub
```

VB6

```
Private Sub BeforeCellEdit(ByVal ItemIndex As Long,ByVal ColIndex As Long,Value As Variant,Cancel As Variant)
End Sub
```

VBA

```
Private Sub BeforeCellEdit(ByVal ItemIndex As Long,ByVal ColIndex As Long,Value As Variant,Cancel As Variant)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex,Value,Cancel
```

Xbas…

```
PROCEDURE OnBeforeCellEdit(oList,ItemIndex,ColIndex,Value,Cancel)
RETURN
```

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

Java…
```
<SCRIPT EVENT="BeforeCellEdit(ItemIndex,ColIndex,Value,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function BeforeCellEdit(ItemIndex,ColIndex,Value,Cancel)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComBeforeCellEdit Integer llItemIndex Integer llColIndex Variant
llValue Variant llCancel
    Forward Send OnComBeforeCellEdit llItemIndex llColIndex llValue llCancel
End_Procedure
```

Visual Objects
```
METHOD OCX_BeforeCellEdit(ItemIndex,ColIndex,Value,Cancel) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_BeforeCellEdit(int _ItemIndex,int _ColIndex,COMVariant /*variant*/
_Value,COMVariant /*variant*/ _Cancel)
{
}
```

XBasic
```
function BeforeCellEdit as v (ItemIndex as N,ColIndex as N,Value as A,Cancel as A)
end function
```

dBASE
```
function nativeObject_BeforeCellEdit(ItemIndex,ColIndex,Value,Cancel)
return
```

The following VB sample disables editing the cells on the second column:

```
Private Sub List1_BeforeCellEdit(ByVal Index As Long, ByVal ColIndex As Long, Value As
Variant, Cancel As Variant)
    Cancel = ColIndex = 1
End Sub
```

The following C++ sample disables editing the cells on the second column:

```
void OnBeforeCellEditList1(long ItemIndex, long ColIndex, VARIANT FAR* Value, VARIANT
FAR* Cancel)
{
   if ( ColIndex == 1 )
   {
      V_VT( Cancel ) = VT_BOOL;
      V_BOOL( Cancel ) = VARIANT_TRUE;
   }
}
```

The following VB.NET sample disables editing the cells on the second column:

```
Private Sub AxList1_BeforeCellEdit(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_BeforeCellEditEvent) Handles AxList1.BeforeCellEdit
   If (e.colIndex = 1) Then
      e.cancel = True
   End If
End Sub
```

The following C# sample disables editing the cells on the second column:

```
private void axList1_BeforeCellEdit(object sender,
AxEXLISTLib._IListEvents_BeforeCellEditEvent e)
{
   if (e.colIndex == 1)
      e.cancel = true;
}
```

The following VFP sample disables editing the cells on the second column:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex, value, cancel

if ( colindex = 1 )
   cancel = .t.
endif
```

event CancelCellEdit (ItemIndex as Long, ColIndex as Long, Reserved as Variant)

Occurs if the edit operation is canceled.

Type	Description
ItemIndex as Long	A long expression that indicates the index of item being edited.
ColIndex as Long	A long expression that indicates the column's index
Reserved as Variant	Contains the caption of the edit control when the user presses the ESC key, or when the user clicks outside the edit field.

The CancelCellEdit event notifies your application that the user cancels the edit operation. The CancelCellEdit event is fired when the user presses ESC key while editing or when the user clicks outside the edit field. The control fires BeforeCellEdit, AfterCellEdit, CancelCellEdit events if the AllowEdit property is True. The AfterCellEdit event notifies your application that the user alters the cell's caption. Use the Edit method to programmatically edit a cell.

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

C#

```
private void CancelCellEdit(object sender,int ItemIndex,int ColIndex,object
Reserved)
{
}
```

VB

```
Private Sub CancelCellEdit(ByVal sender As System.Object,ByVal ItemIndex As
Integer,ByVal ColIndex As Integer,ByVal Reserved As Object) Handles
CancelCellEdit
End Sub
```

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

C#

```
private void CancelCellEdit(object sender,
AxEXLISTLib._IListEvents_CancelCellEditEvent e)
{
}
```

C++

```
void OnCancelCellEdit(long ItemIndex,long ColIndex,VARIANT Reserved)
```

```
{
}
```

C++ Builder

```
void __fastcall CancelCellEdit(TObject *Sender,long ItemIndex,long
ColIndex,Variant Reserved)
{
}
```

Delphi

```
procedure CancelCellEdit(ASender: TObject; ItemIndex : Integer;ColIndex :
Integer;Reserved : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CancelCellEdit(sender: System.Object; e:
AxEXLISTLib._IListEvents_CancelCellEditEvent);
begin
end;
```

Powe…

```
begin event CancelCellEdit(long ItemIndex,long ColIndex,any Reserved)
end event CancelCellEdit
```

VB.NET

```
Private Sub CancelCellEdit(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_CancelCellEditEvent) Handles CancelCellEdit
End Sub
```

VB6

```
Private Sub CancelCellEdit(ByVal ItemIndex As Long,ByVal ColIndex As Long,ByVal
Reserved As Variant)
End Sub
```

VBA

```
Private Sub CancelCellEdit(ByVal ItemIndex As Long,ByVal ColIndex As Long,ByVal
Reserved As Variant)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex,Reserved
```

Xbas…

```
PROCEDURE OnCancelCellEdit(oList,ItemIndex,ColIndex,Reserved)
RETURN
```

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

Java…
```
<SCRIPT EVENT="CancelCellEdit(ItemIndex,ColIndex,Reserved)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function CancelCellEdit(ItemIndex,ColIndex,Reserved)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComCancelCellEdit Integer llItemIndex Integer llColIndex Variant
llReserved
    Forward Send OnComCancelCellEdit llItemIndex llColIndex llReserved
End_Procedure
```

Visual Objects
```
METHOD OCX_CancelCellEdit(ItemIndex,ColIndex,Reserved) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_CancelCellEdit(int _ItemIndex,int _ColIndex,COMVariant _Reserved)
{
}
```

XBasic
```
function CancelCellEdit as v (ItemIndex as N,ColIndex as N,Reserved as A)
end function
```

dBASE
```
function nativeObject_CancelCellEdit(ItemIndex,ColIndex,Reserved)
return
```

The following VB sample changes the cell's caption when the user clicks outside the edit field:

```
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

Private Sub List1_CancelCellEdit(ByVal ItemIndex As Long, ByVal ColIndex As Long, ByVal
Reserved As Variant)
    If Not (GetAsyncKeyState(27) < 0) Then
```

```
    With List1.Items
        .Caption(ItemIndex, ColIndex) = Reserved
    End With
  End If
End Sub
```

The GetAsyncKeyState function determines whether a key is up or down at the time the function is called. The sample changes the selected caption only if the user didn't press ESC key.

The following C++ sample changes the cell's caption when the user clicks outside the edit field:

```
void OnCancelCellEditList1(long ItemIndex, long ColIndex, const VARIANT FAR& Reserved)
{
  if ( ! ( GetAsyncKeyState( VK_ESCAPE ) < 0 ) )
  {
    CString strNewCaption = V2S( (VARIANT*)&Reserved );
    CItems items = m_list.GetItems();
    items.SetCaption( ItemIndex, COleVariant( ColIndex ), COleVariant( strNewCaption ) );
  }
}
```

where the V2S function converts a VARIANT expression to a string expression,

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
  if ( pv )
  {
    if ( pv->vt == VT_ERROR )
      return szDefault;

    COleVariant vt;
    vt.ChangeType( VT_BSTR, pv );
    return V_BSTR( &vt );
  }
  return szDefault;
}
```

event CellButtonClick (ItemIndex as Long, ColIndex as Long)

Fired after the user clicks the cell's button.

Type	Description
ItemIndex as Long	A long value that indicates the index of item being clicked.
ColIndex as Long	A long value that indicates the index of the column.

The CellButtonClick event is fired after the user has released the left mouse button over a cell of button type. Use the CellHasButton property to specify whether a cell is of button type. The CellButtonClick event notifies your application that user presses a cell of button type. Use the Caption property to specify the button's caption.

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

C#
```
private void CellButtonClick(object sender,int ItemIndex,int ColIndex)
{
}
```

VB
```
Private Sub CellButtonClick(ByVal sender As System.Object,ByVal ItemIndex As
Integer,ByVal ColIndex As Integer) Handles CellButtonClick
End Sub
```

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

C#
```
private void CellButtonClick(object sender,
AxEXLISTLib._IListEvents_CellButtonClickEvent e)
{
}
```

C++
```
void OnCellButtonClick(long ItemIndex,long ColIndex)
{
}
```

C++ Builder
```
void __fastcall CellButtonClick(TObject *Sender,long ItemIndex,long ColIndex)
{
}
```

Delphi
```
procedure CellButtonClick(ASender: TObject; ItemIndex : Integer;ColIndex :
Integer);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellButtonClick(sender: System.Object; e:
AxEXLISTLib._IListEvents_CellButtonClickEvent);
begin
end;
```

Powe…

```
begin event CellButtonClick(long ItemIndex,long ColIndex)
end event CellButtonClick
```

VB.NET

```
Private Sub CellButtonClick(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_CellButtonClickEvent) Handles CellButtonClick
End Sub
```

VB6

```
Private Sub CellButtonClick(ByVal ItemIndex As Long,ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub CellButtonClick(ByVal ItemIndex As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex
```

Xbas…

```
PROCEDURE OnCellButtonClick(oList,ItemIndex,ColIndex)
RETURN
```

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

Java…

```
<SCRIPT EVENT="CellButtonClick(ItemIndex,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function CellButtonClick(ItemIndex,ColIndex)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComCellButtonClick Integer llItemIndex Integer llColIndex
    Forward Send OnComCellButtonClick llItemIndex llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_CellButtonClick(ItemIndex,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellButtonClick(int _ItemIndex,int _ColIndex)
{
}
```

XBasic

```
function CellButtonClick as v (ItemIndex as N,ColIndex as N)
end function
```

dBASE

```
function nativeObject_CellButtonClick(ItemIndex,ColIndex)
return
```

The following VB sample prints the caption of the cell being clicked:

```
Private Sub List1_CellButtonClick(ByVal Index As Long, ByVal ColIndex As Long)
    Debug.Print List1.Items.Caption(Index, ColIndex)
End Sub
```

The following C++ sample prints the caption of the cell being clicked:

```
void OnCellButtonClickList1(long ItemIndex, long ColIndex)
{
    CItems items = m_list.GetItems();
    CString strCaption = V2S( &items.GetCaption( ItemIndex, COleVariant( ColIndex ) ) );
    OutputDebugString( strCaption );
}
```

where the V2S function converts a VARIANT expression to a string expression,

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
```

```
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}
```

The following VB.NET sample prints the caption of the cell being clicked:

```
Private Sub AxList1_CellButtonClick(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_CellButtonClickEvent) Handles AxList1.CellButtonClick
    With AxList1.Items
        Debug.Write(.Caption(e.itemIndex, e.colIndex))
    End With
End Sub
```

The following C# sample prints the caption of the cell being clicked:

```
private void axList1_CellButtonClick(object sender,
AxEXLISTLib._IListEvents_CellButtonClickEvent e)
{
    System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(e.itemIndex,
e.colIndex).ToString());
}
```

The following VFP sample prints the caption of the cell being clicked:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex

with thisform.List1.Items
    wait window nowait .Caption(itemindex,colindex)
endwith
```

event CellImageClick (ItemIndex as Long, ColIndex as Long)

Fired after the user clicks the cell's icon.

Type	Description
ItemIndex as Long	A long value that indicates the index of item being clicked.
ColIndex as Long	A long value that indicates the index of column, where the cell's icon is clicked.

The CellImageClick event is fired when user clicks on the cell's image. Use the CellImage property to assign an icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the ItemFromPoint property to determine the index of the icon being clicked, in case the cell displays multiple icons using the CellImages property. Use the CellHasCheckBox or CellHasRadioButton property to assign a check box or a radio button to a cell.

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

C#

```
private void CellImageClick(object sender,int ItemIndex,int ColIndex)
{
}
```

VB

```
Private Sub CellImageClick(ByVal sender As System.Object,ByVal ItemIndex As
Integer,ByVal ColIndex As Integer) Handles CellImageClick
End Sub
```

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

C#

```
private void CellImageClick(object sender,
AxEXLISTLib._IListEvents_CellImageClickEvent e)
{
}
```

C++

```
void OnCellImageClick(long ItemIndex,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall CellImageClick(TObject *Sender,long ItemIndex,long ColIndex)
{
}
```

Delphi

```
procedure CellImageClick(ASender: TObject; ItemIndex : Integer;ColIndex :
Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellImageClick(sender: System.Object; e:
AxEXLISTLib._IListEvents_CellImageClickEvent);
begin
end;
```

Powe…

```
begin event CellImageClick(long ItemIndex,long ColIndex)
end event CellImageClick
```

VB.NET

```
Private Sub CellImageClick(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_CellImageClickEvent) Handles CellImageClick
End Sub
```

VB6

```
Private Sub CellImageClick(ByVal ItemIndex As Long,ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub CellImageClick(ByVal ItemIndex As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex
```

Xbas…

```
PROCEDURE OnCellImageClick(oList,ItemIndex,ColIndex)
RETURN
```

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

Java…

```
<SCRIPT EVENT="CellImageClick(ItemIndex,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function CellImageClick(ItemIndex,ColIndex)
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComCellImageClick Integer llItemIndex Integer llColIndex
    Forward Send OnComCellImageClick llItemIndex llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_CellImageClick(ItemIndex,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellImageClick(int _ItemIndex,int _ColIndex)
{
}
```

XBasic

```
function CellImageClick as v (ItemIndex as N,ColIndex as N)
end function
```

dBASE

```
function nativeObject_CellImageClick(ItemIndex,ColIndex)
return
```

The following VB sample changes the cell's icon when it is clicked:

```
Private Sub List1_CellImageClick(ByVal Index As Long, ByVal ColIndex As Long)
   With List1.Items
      .CellImage(Index, ColIndex) = 1 + (.CellImage(Index, ColIndex)) Mod 2
   End With
End Sub
```

The following C++ sample changes the cell's icon when it is clicked:

```
void OnCellImageClickList1(long ItemIndex, long ColIndex)
{
   COleVariant vtColumn( ColIndex );
   CItems items = m_list.GetItems();
   items.SetCellImage( ItemIndex, vtColumn, ( items.GetCellImage( ItemIndex, vtColumn )
% 2 ) + 1 );
}
```

The following VB.NET sample changes the cell's icon when it is clicked:

```
Private Sub AxList1_CellImageClick(ByVal sender As Object, ByVal e As
```

```
AxEXLISTLib._IListEvents_CellImageClickEvent) Handles AxList1.CellImageClick
    With AxList1.Items
        .CellImage(e.itemIndex, e.colIndex) = .CellImage(e.itemIndex, e.colIndex) Mod 2 + 1
    End With
End Sub
```

The following C# sample changes the cell's icon when it is clicked:

```
private void axList1_CellImageClick(object sender,
AxEXLISTLib._IListEvents_CellImageClickEvent e)
{
    axList1.Items.set_CellImage(e.itemIndex, e.colIndex,
axList1.Items.get_CellImage(e.itemIndex, e.colIndex) % 2 + 1);
}
```

The following VFP sample changes the cell's icon when it is clicked:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex

with thisform.List1.Items
    .CellImage( itemindex, colindex ) = mod( .CellImage( itemindex, colindex ), 2) + 1
endwith
```

event CellStateChanged (ItemIndex as Long, ColIndex as Long)

Fired after cell's state has been changed.

Type	Description
ItemIndex as Long	A long value that indicates the index of the item where the cell's state being changed.
ColIndex as Long	A long value that indicates the column's index.

A cell that contains a radio button or a check box button fires the CellStateChanged event when its state is changed. Use the CellState property to change the cell's state. Use the CellHasRadioButton or CellHasCheckBox property to enable radio or check box button into a cell. Use the CellImage property to display an icon in the cell. Use the CellImages property to display multiple icons in the same cell. Use the CellChecked property to determine the handle of the cell that's checked in a radio group. Use the CellRadioGroup property to radio group cells.

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

C#

```
private void CellStateChanged(object sender,int ItemIndex,int ColIndex)
{
}
```

VB

```
Private Sub CellStateChanged(ByVal sender As System.Object,ByVal ItemIndex As
Integer,ByVal ColIndex As Integer) Handles CellStateChanged
End Sub
```

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

C#

```
private void CellStateChanged(object sender,
AxEXLISTLib._IListEvents_CellStateChangedEvent e)
{
}
```

C++

```
void OnCellStateChanged(long ItemIndex,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall CellStateChanged(TObject *Sender,long ItemIndex,long ColIndex)
{
```

```
}
```

Delphi

```
procedure CellStateChanged(ASender: TObject; ItemIndex : Integer;ColIndex :
Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellStateChanged(sender: System.Object; e:
AxEXLISTLib._IListEvents_CellStateChangedEvent);
begin
end;
```

Powe…

```
begin event CellStateChanged(long ItemIndex,long ColIndex)
end event CellStateChanged
```

VB.NET

```
Private Sub CellStateChanged(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_CellStateChangedEvent) Handles CellStateChanged
End Sub
```

VB6

```
Private Sub CellStateChanged(ByVal ItemIndex As Long,ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub CellStateChanged(ByVal ItemIndex As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex
```

Xbas…

```
PROCEDURE OnCellStateChanged(oList,ItemIndex,ColIndex)
RETURN
```

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

Java…

```
<SCRIPT EVENT="CellStateChanged(ItemIndex,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function CellStateChanged(ItemIndex,ColIndex)
```

```
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComCellStateChanged Integer llItemIndex Integer llColIndex
    Forward Send OnComCellStateChanged llItemIndex llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_CellStateChanged(ItemIndex,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellStateChanged(int _ItemIndex,int _ColIndex)
{
}
```

XBasic

```
function CellStateChanged as v (ItemIndex as N,ColIndex as N)
end function
```

dBASE

```
function nativeObject_CellStateChanged(ItemIndex,ColIndex)
return
```

The following VB sample displays the caption of the cell that's checked into a radio group:

```
Private Sub List1_CellStateChanged(ByVal Index As Long, ByVal ColIndex As Long)
   With List1.Items
      Dim i As Long, c As Long
      .CellChecked 1234, i, c
      Debug.Print .Caption(i, c)
   End With
End Sub
```

Use the CellRadioGroup property to group cells into the same radio group.

The following VB sample groups the cells in the second column:

```
Dim i As Long
With List1.Items
   For i = 0 To .Count - 1
      .CellHasRadioButton(i, 1) = True
```

```
    .CellRadioGroup(i, 1) = 1234
  Next
End With
```

The following VB sample prints the caption of the cell's whose state is changed:

```
Private Sub List1_CellStateChanged(ByVal ItemIndex As Long, ByVal ColIndex As Long)
  With List1.Items
    Debug.Print .Caption(Index, ColIndex)
  End With
End Sub
```

The following C++ sample prints the caption of the cell's whose state is changed:

```
void OnCellStateChangedList1(long ItemIndex, long ColIndex)
{
  if ( IsWindow( m_list.m_hWnd ) )
  {
    CItems items = m_list.GetItems();
    CString strCaption = V2S( &items.GetCaption( ItemIndex, COleVariant( ColIndex ) ) );
    OutputDebugString( strCaption );
  }
}
```

The following VB.NET sample prints the caption of the cell's whose state is changed:

```
Private Sub AxList1_CellStateChanged(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_CellStateChangedEvent) Handles AxList1.CellStateChanged
  With AxList1.Items
    Debug.Write(.Caption(e.itemIndex, e.colIndex))
  End With
End Sub
```

The following C# sample prints the caption of the cell's whose state is changed:

```
private void axList1_CellStateChanged(object sender,
AxEXLISTLib._IListEvents_CellStateChangedEvent e)
{
  System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(e.itemIndex,
```

```
e.colIndex).ToString());
}
```

The following VFP sample prints the caption of the cell's whose state is changed:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex

with thisform.List1.Items
   wait window nowait .Caption(itemindex,colindex)
endwith
```

event CellStateChanging (ItemIndex as Long, ColIndex as Long, NewState as Long)

Fired before cell's state is about to be changed.

Type	Description
ItemIndex as Long	A long expression that indicates the index of the item where the cell's state is about to be changed.
ColIndex as Long	A long expression that indicates the index of the column where the cell's state is changed, or a long expression that indicates the handle of the cell, if the Item parameter is 0.
NewState as Long	A long expression that specifies the new state of the cell (0- unchecked, 1 - checked)

The control fires the CellStateChanging event just before cell's state is about to be changed. For instance, you can prevent changing the cell's state, by calling the NewState = Items.CellState(Item,ColIndex). A cell that contains a radio button or a check box button fires the CellStateChanged event when its state is changed. Use the CellState property to change the cell's state. Use the CellHasRadioButton or CellHasCheckBox property to enable radio or check box button into a cell. Use the Def property to assign check-boxes / radio-buttons for all cells in the column. Use the CellImage property to display an icon in the cell. Use the CellImages property to display multiple icons in the same cell. Use the CellChecked property to determine the handle of the cell that's checked in a radio group. Use the CellRadioGroup property to radio group cells. We would not recommend changing the CellState property during the CellStateChanging event, to prevent recursive calls, instead you can change the NewState parameter which is passed by reference.

Once the user clicks a check-box, radio-button, the control fires the following events:

- CellStateChanging event, where the NewState parameter indicates the new state of the cell's checkbox / radio-button.
- CellStateChanged event notifies your application that the cell's check-box or radio-button has been changed. The CellState property determines the check-box/radio-button state of the cell.

For instance, the following VB sample prevents changing the cell's checkbox/radio-button, when the control's ReadOnly property is set:

```
Private Sub List1_CellStateChanging(ByVal ItemIndex As Long, ByVal ColIndex As Long, NewState As Long)
```

```
    With List1
        If (.ReadOnly) Then
            With .Items
                NewState = .CellState(ItemIndex, ColIndex)
            End With
        End If
    End With
End Sub
```

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

C#

```
private void CellStateChanging(object sender,int   Item,int   ColIndex,ref int
NewState)
{
}
```

VB

```
Private Sub CellStateChanging(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer,ByRef NewState As Integer) Handles
CellStateChanging
End Sub
```

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

C#

```
private void CellStateChanging(object sender,
AxEXLISTLib._IListEvents_CellStateChangingEvent e)
{
}
```

C++

```
void OnCellStateChanging(long   Item,long   ColIndex,long FAR*   NewState)
{
}
```

C++ Builder

```
void __fastcall CellStateChanging(TObject *Sender,long   ItemIndex,long
ColIndex,long *   NewState)
{
}
```

Delphi

```
procedure CellStateChanging(ASender: TObject; ItemIndex : Integer;ColIndex :
Integer;var NewState : Integer);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellStateChanging(sender: System.Object; e:
AxEXLISTLib._IListEvents_CellStateChangingEvent);
begin
end;
```

Powe…

```
begin event CellStateChanging(long  ItemIndex,long  ColIndex,long  NewState)

end event CellStateChanging
```

VB.NET

```
Private Sub CellStateChanging(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_CellStateChangingEvent) Handles CellStateChanging
End Sub
```

VB6

```
Private Sub CellStateChanging(ByVal ItemIndex As Long,ByVal ColIndex As
Long,NewState As Long)
End Sub
```

VBA

```
Private Sub CellStateChanging(ByVal Item As Long,ByVal ColIndex As
Long,NewState As Long)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,NewState
```

Xbas…

```
PROCEDURE OnCellStateChanging(oList,Item,ColIndex,NewState)

RETURN
```

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

Java…

```
<SCRIPT EVENT="CellStateChanging(Item,ColIndex,NewState)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

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

```
Function CellStateChanging(Item,ColIndex,NewState)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComCellStateChanging Integer   llItemIndex Integer   llColIndex
Integer   llNewState
   Forward Send OnComCellStateChanging llItemIndex llColIndex llNewState
End_Procedure
```

Visual Objects

```
METHOD OCX_CellStateChanging(Item,ColIndex,NewState) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellStateChanging(int   _Item,int   _ColIndex,COMVariant /*long*/
_NewState)
{
}
```

XBasic

```
function CellStateChanging as v (ItemIndex  as  N,ColIndex  as  N,NewState  as  N)
end function
```

dBASE

```
function nativeObject_CellStateChanging(ItemIndex,ColIndex,NewState)
return
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the list 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 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(oList)
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 ColumnClick (Column as Column)

Fired after the user clicks on column's header.

Type	Description
Column as Column	A Column object being clicked.

The ColumnClick event is fired when the user clicks the column's header. By default, the control sorts by the column when user clicks the column's header. Use the SortOnClick property to specify the operation that control does when user clicks the column's caption. Use the ColumnFromPoint property to access the column from point. Use the ItemFromPoint property to access the item from point. The control fires Sort method when the control sorts a column. Use the MouseDown or MouseUp event to notify the control when the user clicks the control, including the columns. Use the HeaderVisible property to hide the control's header bar.

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

C#

```
private void ColumnClick(object sender,exontrol.EXLISTLib.Column Column)
{
}
```

VB

```
Private Sub ColumnClick(ByVal sender As System.Object,ByVal Column As
exontrol.EXLISTLib.Column) Handles ColumnClick
End Sub
```

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

C#

```
private void ColumnClick(object sender,
AxEXLISTLib._IListEvents_ColumnClickEvent e)
{
}
```

C++

```
void OnColumnClick(LPDISPATCH Column)
{
}
```

C++ Builder

```
void __fastcall ColumnClick(TObject *Sender,Exlistlib_tlb::IColumn *Column)
{
}
```

Delphi

```
procedure ColumnClick(ASender: TObject; Column : IColumn);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ColumnClick(sender: System.Object; e:
AxEXLISTLib._IListEvents_ColumnClickEvent);
begin
end;
```

Powe…

```
begin event ColumnClick(oleobject Column)
end event ColumnClick
```

VB.NET

```
Private Sub ColumnClick(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_ColumnClickEvent) Handles ColumnClick
End Sub
```

VB6

```
Private Sub ColumnClick(ByVal Column As EXLISTLibCtl.IColumn)
End Sub
```

VBA

```
Private Sub ColumnClick(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnColumnClick(oList,Column)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComColumnClick Variant llColumn
   Forward Send OnComColumnClick llColumn
End_Procedure
```

Visual Objects

```
METHOD OCX_ColumnClick(Column) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ColumnClick(COM _Column)
{
}
```

XBasic

```
function ColumnClick as v (Column as OLE::Exontrol.List.1::IColumn)
end function
```

dBASE

```
function nativeObject_ColumnClick(Column)
return
```

The following VB sample displays the caption of the column being clicked:

```
Private Sub List1_ColumnClick(ByVal Column As EXLISTLibCtl.IColumn)
   Debug.Print Column.Caption
End Sub
```

The following C++ sample displays the caption of the column being clicked:

```
#include "Column.h"
void OnColumnClickList1(LPDISPATCH Column)
{
   CColumn column( Column );
   column.m_bAutoRelease = FALSE;
   MessageBox( column.GetCaption() );

}
```

The following VB.NET sample displays the caption of the column being clicked:

```
Private Sub AxList1_ColumnClick(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_ColumnClickEvent) Handles AxList1.ColumnClick
   MessageBox.Show(e.column.Caption)
End Sub
```

The following C# sample displays the caption of the column being clicked:

```
private void axList1_ColumnClick(object sender,
AxEXLISTLib._IListEvents_ColumnClickEvent e)
{
   MessageBox.Show( e.column.Caption );
}
```

The following VFP sample displays the caption of the column being clicked:

```
*** ActiveX Control Event ***
LPARAMETERS column

with column
   wait window nowait .Caption
endwith
```

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

Occurs when the user presses and releases a mouse button and then presses and releases it again over an object.

Type	Description
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The DblClick event is fired when user double clicks the control. Use the ItemFromPoint method to determine the cell over the cursor. Use the ColumnFromPoint property to get the column from point. The MouseDown event notifies your application that user clicks the control.

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, AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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.List.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.List.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the caption of the cell being double clicked:

```
Private Sub List1_DblClick(Shift As Integer, X As Single, Y As Single)
  Dim c As Long, i As Long, hit As HitTestInfoEnum
  With List1
    i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If (i >= 0) Then
      If (c >= 0) Then
        Debug.Print "DblClk on " & .Items.Caption(i, c)
      End If
    End If
  End With
End Sub
```

The following C++ sample displays the caption of the cell being double clicked:

```
void OnDblClickList1(short Shift, long X, long Y)
{
   long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
   if ( i >= 0 )
   {
      CItems items = m_list.GetItems();
      CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
      OutputDebugString( strCaption );
   }
}
```

The following VB.NET sample displays the caption of the cell being double clicked:

```
Private Sub AxList1_DblClick(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_DblClickEvent) Handles AxList1.DblClick
   Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
   Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
   If (i >= 0) Then
      With AxList1.Items
         Debug.Write(.Caption(i, c))
      End With
   End If
End Sub
```

The following C# sample displays the caption of the cell being double clicked:

```
private void axList1_DblClick(object sender, AxEXLISTLib._IListEvents_DblClickEvent e)
{
   EXLISTLib.HitTestInfoEnum hit;
   int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit );
   if (i >= 0)
   {
      System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(i, c).ToString());
   }
}
```

The following VFP sample displays the caption of the cell being double clicked:

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

local c, i, hit
With thisform.List1
   c = 0
   hit = 0
   i = .ItemFromPoint(x, y, @c, @hit)
   If (i >= 0)
      wait window nowait .Items.Caption(i, c)
   EndIf
EndWith
```

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 exgantt1.EventParam(-2).toString();
}
```

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

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

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

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

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

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   ;
   if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem as HITEM, Cancel as Boolean) */
      exgantt1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

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

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

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   ;
   if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem as HITEM, Cancel as Boolean) */
      if ( !exgantt1.Items().EnableItem( exgantt1.EventParam( 2 /*NewItem*/ ) ) )
         exgantt1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

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

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

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

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

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

C#
```
private void Event(object sender, AxEXLISTLib._IListEvents_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: AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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 FilterChange ()

Occurs when filter was changed.

Type	Description

Use the FilterChange event to notify your application that the control's filter is changed. The FilterChanging event occurs just before applying the filter. Use the Filter and FilterType properties to retrieve the column's filter string, if case, and the column's filter type. The ApplyFilter and ClearFilter methods fire the FilterChange event. Use the DisplayFilterButton property to add a filter bar button to the column's caption. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar.

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

C#

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

VB

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

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

C#

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

C++

```
void OnFilterChange()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event FilterChange()
end event FilterChange
```

VB.NET

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

VB6

```
Private Sub FilterChange()
End Sub
```

VBA

```
Private Sub FilterChange()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnFilterChange(oList)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComFilterChange
   Forward Send OnComFilterChange
```

```
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_FilterChange()
{
}
```

XBasic

```
function FilterChange as v ()
end function
```

dBASE

```
function nativeObject_FilterChange()
return
```

event FilterChanging ()

Notifies your application that the filter is about to change.

Type	Description

The FilterChanging event occurs just before applying the filter. The FilterChange event occurs once the filter is applied, so the list gets filtered. Use the Filter and FilterType properties to retrieve the column's filter string, if case, and the column's filter type. The ApplyFilter and ClearFilter methods fire the FilterChange event. Use the DisplayFilterButton property to add a filter bar button to the column's caption. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar. For instance, you can use the FilterChanging event to start a timer, and count the time to get the filter applied, when the FilterChange event is fired.

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

C#

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

VB

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

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

C#

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

C++

```
void OnFilterChanging()
{
}
```

C++ Builder

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

Delphi

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

```
end;
```

Delphi 8 (.NET only)

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

Powe…

```
begin event FilterChanging()
end event FilterChanging
```

VB.NET

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

VB6

```
Private Sub FilterChanging()
End Sub
```

VBA

```
Private Sub FilterChanging()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnFilterChanging(oList)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComFilterChanging
   Forward Send OnComFilterChanging
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_FilterChanging()
{
}
```

XBasic

```
function FilterChanging as v ()
end function
```

dBASE

```
function nativeObject_FilterChanging()
return
```

event FormatColumn (ItemIndex as Long, ColIndex as Long, Value as Variant)

Fired when a cell requires to format its caption.

Type	Description
ItemIndex as Long	A long value that indicates the index of item being formatted.
ColIndex as Long	A long value that indicates the column's index.
Value as Variant	A VARIANT value that indicates the value that being displayed. Initially, the Value parameter indicates the the Caption value.

Use the FormatColumn event to display a string different than the Caption property. The FormatColumn event is fired only if the FireFormatColumn property of the Column is True. The FormatColumn event lets the user to provide the cell's caption before it is displayed on the control's list. For instance, the FormatColumn event is useful when the column cells contains prices(numbers), and you want to display that column formatted as currency, like $150 instead 150. Also, using the FormatColumn event, you can display the result of some operations within an item, such of totals. *Newer versions of the component provides the FormatColumn property that helps formatting a cell using the several predefined functions without using the control's event FormatColumn.*

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

C#
```
private void FormatColumn(object sender,int ItemIndex,int ColIndex,ref object Value)
{
}
```

VB
```
Private Sub FormatColumn(ByVal sender As System.Object,ByVal ItemIndex As Integer,ByVal ColIndex As Integer,ByRef Value As Object) Handles FormatColumn
End Sub
```

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

C#
```
private void FormatColumn(object sender, AxEXLISTLib._IListEvents_FormatColumnEvent e)
{
}
```

C++

```
void OnFormatColumn(long ItemIndex,long ColIndex,VARIANT FAR* Value)
{
}
```

C++ Builder

```
void __fastcall FormatColumn(TObject *Sender,long ItemIndex,long
ColIndex,Variant * Value)
{
}
```

Delphi

```
procedure FormatColumn(ASender: TObject; ItemIndex : Integer;ColIndex :
Integer;var Value : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure FormatColumn(sender: System.Object; e:
AxEXLISTLib._IListEvents_FormatColumnEvent);
begin
end;
```

Powe…

```
begin event FormatColumn(long ItemIndex,long ColIndex,any Value)
end event FormatColumn
```

VB.NET

```
Private Sub FormatColumn(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_FormatColumnEvent) Handles FormatColumn
End Sub
```

VB6

```
Private Sub FormatColumn(ByVal ItemIndex As Long,ByVal ColIndex As Long,Value
As Variant)
End Sub
```

VBA

```
Private Sub FormatColumn(ByVal ItemIndex As Long,ByVal ColIndex As Long,Value
As Variant)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex,Value
```

Xbas…

```
PROCEDURE OnFormatColumn(oList,ItemIndex,ColIndex,Value)
```

```
RETURN
```

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

Java...
```
<SCRIPT EVENT="FormatColumn(ItemIndex,ColIndex,Value)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc...
```
<SCRIPT LANGUAGE="VBScript">
Function FormatColumn(ItemIndex,ColIndex,Value)
End Function
</SCRIPT>
```

Visual Data...
```
Procedure OnComFormatColumn Integer llItemIndex Integer llColIndex Variant
llValue
   Forward Send OnComFormatColumn llItemIndex llColIndex llValue
End_Procedure
```

Visual Objects
```
METHOD OCX_FormatColumn(ItemIndex,ColIndex,Value) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_FormatColumn(int _ItemIndex,int _ColIndex,COMVariant /*variant*/
_Value)
{
}
```

XBasic
```
function FormatColumn as v (ItemIndex as N,ColIndex as N,Value as A)
end function
```

dBASE
```
function nativeObject_FormatColumn(ItemIndex,ColIndex,Value)
return
```

Before running any of the following samples, please make sure that the control contains more than 3 columns, and the third column has the FireFormatColumn property on True. The following VB sample displays the sum of the first two cells, and put the result on the third one:

```
Private Sub List1_FormatColumn(ByVal ItemIndex As Long, ByVal ColIndex As Long, Value
```

```
As Variant)
On Error Resume Next
   With List1.Items
      Value = Int(.Caption(ItemIndex, 0)) + Int(.Caption(ItemIndex, 1))
   End With
End Sub
```

The following VB sample displays long date format, using the FormatDateTime function:

```
Private Sub List1_FormatColumn(ByVal ItemIndex As Long, ByVal ColIndex As Long, Value
As Variant)
On Error Resume Next
   Value = FormatDateTime(Value, vbLongDate)
End Sub
```

The following C++ sample displays the sum of the first two cells, and put the result on the third one:

```
void OnFormatColumnList1(long ItemIndex, long ColIndex, VARIANT FAR* Value)
{
   CItems items = m_list.GetItems();
   long newValue = V2I( &items.GetCaption( ItemIndex, COleVariant( long(0) ) ) );
   newValue += V2I( &items.GetCaption( ItemIndex, COleVariant( long(1) ) ) );
   V_VT( Value ) = VT_I4;
   V_I4( Value ) = newValue;
}
```

where the V2I function converts a VARIANT value to a long expression,

```
static long V2I( VARIANT* pv, long nDefault = 0 )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return nDefault;

      COleVariant vt;
      vt.ChangeType( VT_I4, pv );
      return V_I4( &vt );
```

```
    }
    return nDefault;
}
```

The following C++ sample displays long date format:

```
void OnFormatColumnList1(long ItemIndex, long ColIndex, VARIANT FAR* Value)
{
    COleDateTime date( *Value );
    COleVariant vtNewValue( date.Format( _T("%A, %B %d, %Y") ) );
    VariantCopy( Value, vtNewValue );
}
```

The following VB.NET sample displays the sum of the first two cells, and put the result on the third one:

```
Private Sub AxList1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_FormatColumnEvent) Handles AxList1.FormatColumn
    With AxList1.Items
        Dim newValue As Integer = Integer.Parse(.Caption(e.itemIndex, 0),
Globalization.NumberStyles.Any)
        newValue = newValue + Integer.Parse(.Caption(e.itemIndex, 1),
Globalization.NumberStyles.Any)
        e.value = newValue
    End With
End Sub
```

The following VB.NET sample displays long date format:

```
Private Sub AxList1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_FormatColumnEvent) Handles AxList1.FormatColumn
    e.value = DateTime.Parse(e.value).ToLongDateString()
End Sub
```

The following C# sample displays the sum of the first two cells, and put the result on the third one:

```
private void axList1_FormatColumn(object sender,
AxEXLISTLib._IListEvents_FormatColumnEvent e)
```

```
{
   int newValue = int.Parse(axList1.Items.get_Caption(e.itemIndex, 0).ToString());
   newValue += int.Parse(axList1.Items.get_Caption(e.itemIndex, 1).ToString());
   e.value = newValue;
}
```

The following C# sample displays long date format:

```
private void axList1_FormatColumn(object sender,
AxEXLISTLib._IListEvents_FormatColumnEvent e)
{
   e.value = DateTime.Parse(e.value.ToString()).ToLongDateString();
}
```

The following VFP sample displays the sum of the first two cells, and put the result on the third one:

```
*** ActiveX Control Event ***
LPARAMETERS itemindex, colindex, value

with thisform.List1.Items
   value = .Caption(itemindex,0) + .Caption(itemindex,1)
endwith
```

event KeyDown (KeyCode as Integer, Shift as Integer)

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

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

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
```

In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

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

C#

```
private void KeyDown(object sender,ref short KeyCode,short Shift)
{
}
```

VB

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

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

C#

```
private void KeyDownEvent(object sender, AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,KeyCode,Shift)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

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

Visual Objects

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

X++

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

XBasic

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

dBASE

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

event KeyPress (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	An integer that returns a standard numeric ANSI keycode.

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

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

C#

```
private void KeyPress(object sender,ref short KeyAscii)
{
}
```

VB

```
Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

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

C#

```
private void KeyPressEvent(object sender, AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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, AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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 LayoutChanged ()

Occurs when column's position or column's size is changed.

Type	Description

The LayoutChanged event is fired each time when the user resizes a column, or drags the column to a new position. Use the LayoutChanged event to notify your application that the columns position or size is changed. Use the LayoutChanged event to save the columns position and size for future use. Use the Width property to retrieve the column's width. Use the Position property to retrieve the column's position. The Visible property specifies whether a column is shown or hidden. Use the ColumnAutoResize property to specify whether the visible columns fit the control's client area.

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

C#

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

VB

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

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

C#

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

C++

```
void OnLayoutChanged()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event LayoutChanged()
end event LayoutChanged
```

VB.NET

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

VB6

```
Private Sub LayoutChanged()
End Sub
```

VBA

```
Private Sub LayoutChanged()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnLayoutChanged(oList)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComLayoutChanged
    Forward Send OnComLayoutChanged
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_LayoutChanged()
{
}
```

XBasic

```
function LayoutChanged as v ()
end function
```

dBASE

```
function nativeObject_LayoutChanged()
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 ItemFromPoint property to retrieve the cell over the cursor. Use the ColumnFromPoint property to get the column from point.

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,
AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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.List.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.List.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the cell's caption from the cursor:

```
Private Sub List1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim c As Long, i As Long, hit As HitTestInfoEnum
   With List1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
```

```
        If (i >= 0) Then
            If (c >= 0) Then
                Debug.Print .Items.Caption(i, c)
            End If
        End If
    End With
End Sub
```

The getCellPos function determines the client coordinates of the cell:

```
Private Sub getCellPos(ByVal l As EXLISTLibCtl.List, ByVal nItem As Long, ByVal nColumn As
Long, x As Long, y As Long)
    x = -l.ScrollPos(False)
    With l
        Dim c As EXLISTLibCtl.Column
        For Each c In .Columns
            If (c.Visible) Then
                If (c.Position < .Columns(nColumn).Position) Then
                    x = x + c.Width
                End If
            End If
        Next
        y = 0
        If (.HeaderVisible) Then
            y = y + .HeaderHeight
        End If
        With .Items
            Dim i As Long
            i = .FirstVisibleItem
            While Not (i = nItem) And (i >= 0)
                y = y + .ItemHeight(i)
                i = .NextVisibleItem(i)
            Wend
        End With
    End With
End Sub
```

The getCellPos method gets the x, y client coordinates of the cell (nItem, nColumn). The nItem indicates the index of the item, and the nColumn indicates the index of the column. Use the ClientToScreen API function to convert the client coordinates to screen coordinates like bellow:

```
Private Type POINTAPI
    x As Long
    y As Long
End Type
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As Long, lpPoint As
POINTAPI) As Long
```

In the following MouseDown handler the ItemFromPoint method determines the cell from the cursor. The sample displays an exPopupMenu control at the beginning of the cell, when user right clicks the cell:

```
Private Sub List1_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
  If Button = 2 Then
    With List1
      Dim i As Long, c As Long, hit As HitTestInfoEnum
      i = .ItemFromPoint(x / Screen.TwipsPerPixelX, y / Screen.TwipsPerPixelY, c, hit)
      If (i >= 0) Then
        ' Selects the item when user does a right click
        List1.Items.SelectItem(i) = True
        ' Gets the client coordinates of the cell
        Dim xCell As Long, yCell As Long
        getCellPos List1, i, c, xCell, yCell
        ' Converts the client coordinates to the screen coordinates
        Dim p As POINTAPI
        p.x = xCell
        p.y = yCell
        ClientToScreen List1.hwnd, p
        ' Displays the exPopupMenu control at specified position
        PopupMenu1.HAlign = EXPOPUPMENULibCtl.exLeft
        Debug.Print "You have selected " & PopupMenu1.Show(p.x, p.y)
      End If
    End With
  End If
```

```
End Sub
```

The following C++ sample displays the caption of the cell being clicked:

```
void OnMouseDownList1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
   if ( i >= 0 )
   {
      CItems items = m_list.GetItems();
      CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
      OutputDebugString( strCaption );
   }
}
```

The following C++ sample displays the caption of the cell being clicked:

```
void OnMouseDownList1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
   if ( i >= 0 )
   {
      CItems items = m_list.GetItems();
      CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
      OutputDebugString( strCaption );
   }
}
```


The following VB.NET sample displays the caption of the cell being clicked:

```
Private Sub AxList1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseDownEvent) Handles AxList1.MouseDownEvent
   Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
   Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
   If (i >= 0) Then
      With AxList1.Items
         Debug.Write(.Caption(i, c))
      End With
   End If
```

```
End Sub
```

The following C# sample displays the caption of the cell being clicked:

```
private void axList1_MouseDownEvent(object sender,
AxEXLISTLib._IListEvents_MouseDownEvent e)
{
   EXLISTLib.HitTestInfoEnum hit;
   int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
   if (i >= 0)
   {
      System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(i, c).ToString());
   }
}
```

The following VFP sample displays the caption of the cell being clicked:

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

local c, i, hit
With thisform.List1
   c = 0
   hit = 0
   i = .ItemFromPoint(x, y, @c, @hit)
   If (i >= 0)
      wait window nowait .Items.Caption(i, c)
   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. Gets which mouse button was pressed as 1 for Left Mouse Button, 2 for Right Mouse Button and 4 for Middle Mouse Button.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The 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. Use the ItemFromPoint property to retrieve the cell over the cursor. Use the ColumnFromPoint property to get the column from point:

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,
AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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.List.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.List.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the cell over the cursor:

```
Private Sub List1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim c As Long, i As Long, hit As HitTestInfoEnum
```

```
    With List1
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If (i >= 0) Then
            If (c >= 0) Then
                Debug.Print .Items.Caption(i, c)
            End If
        End If
    End With
End Sub
```

The following C++ sample displays the cell over the cursor:

```
void OnMouseMoveList1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
    if ( i >= 0 )
    {
        CItems items = m_list.GetItems();
        CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
        OutputDebugString( strCaption );
    }
}
```

The following VB.NET sample displays the cell over the cursor:

```
Private Sub AxList1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseMoveEvent) Handles AxList1.MouseMoveEvent
    Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
    Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
    If (i >= 0) Then
        With AxList1.Items
            Debug.Write(.Caption(i, c))
        End With
    End If
End Sub
```

The following C# sample displays the cell over the cursor:

```
private void axList1_MouseMoveEvent(object sender,
```

```
AxEXLISTLib._IListEvents_MouseMoveEvent e)
{
    EXLISTLib.HitTestInfoEnum hit;
    int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i >= 0)
    {
        System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(i, c).ToString());
    }
}
```

The following VFP sample displays the cell over the cursor:

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

local c, i, hit
With thisform.List1
    c = 0
    hit = 0
    i = .ItemFromPoint(x, y, @c, @hit)
    If (i >= 0)
        wait window nowait .Items.Caption(i, c)
    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 ItemFromPoint property to retrieve the cell over the cursor. Use the ColumnFromPoint property to get the column from point.

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,
AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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.List.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.List.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the cell over the cursor:

```
Private Sub List1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim c As Long, i As Long, hit As HitTestInfoEnum
   With List1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
```

```
    If (i >= 0) Then
        If (c >= 0) Then
            Debug.Print .Items.Caption(i, c)
        End If
    End If
  End With
End Sub
```

The following C++ sample displays the cell over the cursor:

```
void OnMouseUpList1(short Button, short Shift, long X, long Y)
{
  long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
  if ( i >= 0 )
  {
    CItems items = m_list.GetItems();
    CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
    OutputDebugString( strCaption );
  }
}
```

The following VB.NET sample displays the cell over the cursor:

```
Private Sub AxList1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseUpEvent) Handles AxList1.MouseUpEvent
  Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
  Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
  If (i >= 0) Then
    With AxList1.Items
      Debug.Write(.Caption(i, c))
    End With
  End If
End Sub
```

The following C# sample displays the cell over the cursor:

```
private void axList1_MouseUpEvent(object sender,
AxEXLISTLib._IListEvents_MouseUpEvent e)
{
```

```
    EXLISTLib.HitTestInfoEnum hit;
    int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i >= 0)
    {
        System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(i, c).ToString());
    }
}
```

The following VFP sample displays the cell over the cursor:

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

local c, i, hit
With thisform.List1
    c = 0
    hit = 0
    i = .ItemFromPoint(x, y, @c, @hit)
    If (i >= 0)
        wait window nowait .Items.Caption(i, c)
    EndIf
EndWith
```

event OffsetChanged (Horizontal as Boolean, NewVal as Long)

Occurs when the scroll position is changed.

Type	Description
Horizontal as Boolean	A boolean expression that indicates whether the horizontal scroll bar is changed. If the Horizontal parameter is True the position of the horizontal scroll bar is changed. If the Horizontal parameter is False, the position of the vertical scroll bar is changed.
NewVal as Long	A long value that indicates the new scroll bar value.

The event OffsetChanged is not fired if the control's list has no scroll bars. The control adds automatically scroll bar to its list based on the control's content. Use the OffsetChanged event to be notified when one of the control's scroll bars has been changed. Use the ScrollBars property to specify the type of scroll bars that control has. Use the ScrollPos property to scroll the control's content.

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

C#

```
private void OffsetChanged(object sender,bool Horizontal,int NewVal)
{
}
```

VB

```
Private Sub OffsetChanged(ByVal sender As System.Object,ByVal Horizontal As
Boolean,ByVal NewVal As Integer) Handles OffsetChanged
End Sub
```

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

C#

```
private void OffsetChanged(object sender,
AxEXLISTLib._IListEvents_OffsetChangedEvent e)
{
}
```

C++

```
void OnOffsetChanged(BOOL Horizontal,long NewVal)
{
}
```

C++ Builder

```
void __fastcall OffsetChanged(TObject *Sender,VARIANT_BOOL Horizontal,long
NewVal)
```

```
{
}
```

Delphi

```
procedure OffsetChanged(ASender: TObject; Horizontal : WordBool;NewVal :
Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OffsetChanged(sender: System.Object; e:
AxEXLISTLib._IListEvents_OffsetChangedEvent);
begin
end;
```

Powe…

```
begin event OffsetChanged(boolean Horizontal,long NewVal)
end event OffsetChanged
```

VB.NET

```
Private Sub OffsetChanged(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_OffsetChangedEvent) Handles OffsetChanged
End Sub
```

VB6

```
Private Sub OffsetChanged(ByVal Horizontal As Boolean,ByVal NewVal As Long)
End Sub
```

VBA

```
Private Sub OffsetChanged(ByVal Horizontal As Boolean,ByVal NewVal As Long)
End Sub
```

VFP

```
LPARAMETERS Horizontal,NewVal
```

Xbas…

```
PROCEDURE OnOffsetChanged(oList,Horizontal,NewVal)
RETURN
```

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

Java…

```
<SCRIPT EVENT="OffsetChanged(Horizontal,NewVal)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

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

```
Function OffsetChanged(Horizontal,NewVal)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOffsetChanged Boolean llHorizontal Integer llNewVal
   Forward Send OnComOffsetChanged llHorizontal llNewVal
End_Procedure
```

Visual Objects

```
METHOD OCX_OffsetChanged(Horizontal,NewVal) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OffsetChanged(boolean _Horizontal,int _NewVal)
{
}
```

XBasic

```
function OffsetChanged as v (Horizontal as L,NewVal as N)
end function
```

dBASE

```
function nativeObject_OffsetChanged(Horizontal,NewVal)
return
```

The following VB sample displays the new scroll position when user scrolls horizontally the control:

```
Private Sub List1_OffsetChanged(ByVal Horizontal As Boolean, ByVal NewVal As Long)
   If (Horizontal) Then
      Debug.Print "The horizontal scroll bar has been moved to " & NewVal
   End If
End Sub
```

The following C++ sample displays the new scroll position when the user scrolls vertically the control:

```
void OnOffsetChangedList1(BOOL Horizontal, long NewVal)
{
   if ( !Horizontal )
   {
      CString strFormat;
```

```
        strFormat.Format( "NewPos = %i\n", NewVal );
        OutputDebugString( strFormat );
    }
}
```

The following VB.NET sample displays the new scroll position when the user scrolls vertically the control:

```
Private Sub AxList1_OffsetChanged(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_OffsetChangedEvent) Handles AxList1.OffsetChanged
    If (Not e.horizontal) Then
        Debug.WriteLine(e.newVal)
    End If
End Sub
```

The following C# sample displays the new scroll position when the user scrolls vertically the control:

```
private void axList1_OffsetChanged(object sender,
AxEXLISTLib._IListEvents_OffsetChangedEvent e)
{
    if (!e.horizontal)
        System.Diagnostics.Debug.WriteLine(e.newVal);
}
```

The following VFP sample displays the new scroll position when the user scrolls vertically the control:

```
*** ActiveX Control Event ***
LPARAMETERS horizontal, newval

if ( 0 # horizontal )
    wait window nowait str( newval )
endif
```

event OLECompleteDrag (Effect as Long)

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled

Type	Description
Effect as Long	A long set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another

The OLECompleteDrag event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the effect parameter of the OLEDragDrop event. Based on this, the source can then determine the appropriate action it needs to take. For example, if the object was moved into the target (exDropEffectMove), the source needs to delete the object from itself after the move. The control supports only manual OLE drag and drop events. In order to enable OLE drag and drop feature into control you have to check the OLEDropMode and OLEDrag properties.

The settings for effect are:

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

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

C#
```
// OLECompleteDrag event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLECompleteDrag event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

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

C#
```
private void OLECompleteDrag(object sender,
AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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 Remarks.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
Shift as Integer	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

In the /NET Assembly, you have to use the DragDrop event as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

The OLEDragDrop event is fired when the user has dropped files or clipboard information into the control. Use the OLEDropMode property on exOLEDropManual to enable OLE drop and drop support. Use the ItemFromPoint property to get the item from point. Use the ColumnFromPoint property to get the column from point. Use the Add method to add a new item to the control. Use the ItemPosition property to specify the item's position.

The settings for Effect are:

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

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

C#
```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

VB
```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

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

C#
```
private void OLEDragDrop(object sender,
AxEXLISTLib._IListEvents_OLEDragDropEvent e)
{
}
```

C++
```
void OnOLEDragDrop(LPDISPATCH Data,long FAR* Effect,short Button,short
Shift,long X,long Y)
{
}
```

C++ Builder
```
void __fastcall OLEDragDrop(TObject *Sender,Exlistlib_tlb::IExDataObject
*Data,long * Effect,short Button,short Shift,int X,int Y)
```

```
{
}
```

Delphi

```
procedure OLEDragDrop(ASender: TObject; Data : IExDataObject;var Effect :
Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OLEDragDrop(sender: System.Object; e:
AxEXLISTLib._IListEvents_OLEDragDropEvent);
begin
end;
```

Powe…

```
begin event OLEDragDrop(oleobject Data,long Effect,integer Button,integer
Shift,long X,long Y)
end event OLEDragDrop
```

VB.NET

```
Private Sub OLEDragDrop(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_OLEDragDropEvent) Handles OLEDragDrop
End Sub
```

VB6

```
Private Sub OLEDragDrop(ByVal Data As EXLISTLibCtl.IExDataObject,Effect As
Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As
Single)
End Sub
```

VBA

```
Private Sub OLEDragDrop(ByVal Data As Object,Effect As Long,ByVal Button As
Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Data,Effect,Button,Shift,X,Y
```

Xbas…

```
PROCEDURE OnOLEDragDrop(oList,Data,Effect,Button,Shift,X,Y)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComOLEDragDrop Variant llData Integer llEffect Short llButton
Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY
   Forward Send OnComOLEDragDrop llData llEffect llButton llShift llX llY
End_Procedure
```

Visual Objects

```
METHOD OCX_OLEDragDrop(Data,Effect,Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

XBasic

```
function OLEDragDrop as v (Data as OLE::Exontrol.List.1::IExDataObject,Effect as
N,Button as N,Shift as N,X as OLE::Exontrol.List.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.List.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_OLEDragDrop(Data,Effect,Button,Shift,X,Y)
return
```

The idea of drag and drop in exList control is the same as in other controls. To start accepting drag and drop sources the exList control should have the OLEDropMode to exOLEDropManual. Once that is is set, the exList starts accepting any drag and drop sources.

The first step is if you want to be able to drag items from your exList control to other controls the idea is to handle the OLE_StartDrag event. The event passes an object ExDataObject (Data) as argument. The Data and AllowedEffects should be set inside of handler event. Here's a sample:

```
Private Sub List1_OLEStartDrag(ByVal Data As EXLISTLibCtl.IExDataObject, AllowedEffects
```

```
As Long)
   Dim i As Long
   Dim str As String
   With List1.Items
      For i = 0 To .SelectCount - 1
         str = str + .Caption(.SelectedItem(i), 0)
         str = str + vbCrLf
      Next
      AllowedEffects = 1
      Data.SetData str, exCFText
   End With
End Sub
```

What the above code does? It takes each selected item from the exList source, and builds a string that will be passed to the clipboard object Data as text string. So, the clipboard data will contains text, so you will be able to drag the items to a text control. Of course the target controls should have enabled the OLE drag and drop features, and it depends on each control. The AllowedEffects = 1 specifies the type of cursor used in drag and drop operations. The value should be a combination of one of the exOLEDropEffectEnum type. Please check your environment browsed for the all possible values: 1 - Copy, 2 - Move, 0 - None, So, AllowedEffects = 1 specifies the "copy" cursor. If you don't set the AllowedEffects parameter, the cursor is by default None. There is no rule how you have to store the data into the clipboard Data object. It depends how you would like to do the drag and drop operations works.

The second step is accepting OLE drag and drop source objects. That means, if you would like to let your control accept drag and drop objects, you have to handle the OLEDragDrop event. It passes as argument an object Data that stores the drag and drop information. The next sample shows how to take each line saved into the data object and add for each line found a new item into your exList control:

```
Private Sub List2_OLEDragDrop(ByVal Data As Object, Effect As Long, ByVal Button As
Integer, ByVal Shift As Integer, ByVal X As Long, ByVal Y As Long)
   Dim str As String
   str = Data.GetData(exCFText)
   if Not (Right(str, Len(vbCrLf)) = vbCrLf) Then
      str = str + vbCrLf
   End If
   With List2
      .BeginUpdate
```

```
        Dim c As Long, i As Long, n1 As Long, n2 As Long, nPos As Long
        While .Items.SelectCount <> 0
            .Items.SelectItem(.Items.SelectedItem(0)) = False
        Wend
        i = .ItemFromPoint(X / 15, Y / 15, c)
        If (i >= 0) Then
            nPos = .Items.ItemPosition(i)
        End If
        Debug.Print "Index = " & i
        n1 = 1
        n2 = InStr(n1, str, vbCrLf)
        While (n2 > 0)
            Dim n As Long
            n = .Items.Add(Mid(str, n1, n2 - n1))
            If .Items.Caption(n, 0) = "" Then
                .Items.ItemBreak(n) = DotLine
            End If
            If (i >= 0) Then
                .Items.ItemPosition(n) = nPos + 1
                nPos = nPos + 1
            End If
            .Items.SelectItem(n) = True
            n1 = n2 + Len(vbCrLf)
            n2 = InStr(n1, str, vbCrLf)
        Wend
        .EndUpdate
    End With
End Sub
```

The following VB sample adds a new item when the user drags a file (Open the Windows Explorer, click and drag a file to the control) :

```
Private Sub List1_OLEDragDrop(ByVal Data As EXLISTLibCtl.IExDataObject, Effect As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    If Data.GetFormat(exCFFiles) Then
        Data.GetData (exCFFiles)
        Dim strFile As String
        strFile = Data.Files(0)
```

```
        'Adds a new item to the control
        List1.Visible = False
        With List1
        .BeginUpdate
            Dim i As Long
            i = .Items.Add(strFile)
            .Items.EnsureVisibleItem i
        .EndUpdate
        End With
        List1.Visible = True
    End If
End Sub
```

The following VC sample inserts a child item for each file that user drags:

```
#import <exlist.dll> rename( "GetItems", "exGetItems" )

#include "Items.h"
void OnOLEDragDropList1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
    EXLISTLib::IExDataObjectPtr spData( Data );
    if ( spData != NULL )
        if ( spData->GetFormat( EXLISTLib::exCFFiles ) )
        {
            CItems items = m_list.GetItems();
            long c = 0, h = 0, iFrom = m_list.GetItemFromPoint( X, Y, &c, &h );
            EXLISTLib::IExDataObjectFilesPtr spFiles( spData->Files );
            if ( spFiles->Count > 0 )
            {
                m_list.BeginUpdate();
                COleVariant vtMissing; vtMissing.vt = VT_ERROR;
                for ( long i = 0; i < spFiles->Count; i++ )
                {
                    long h = items.Add( COleVariant( spFiles->GetItem( i ).operator const char *() )
);
                    if ( i >= 0 )
```

```
                items.SetItemPosition( h, items.GetItemPosition(iFrom) );
            }
            m_list.EndUpdate();
        }
    }
}
```

The #import statement imports definition for the ExDataObject and ExDataObjectFiles objects. If the exlist.dll file is located in another folder than the system folder, the path to the file must be specified. The sample gets the item where the files were dragged and insert all files in that position, as child items, if case.

The following VB.NET sample inserts a child item for each file that user drags:

```
Private Sub AxList1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_OLEDragDropEvent) Handles AxList1.OLEDragDrop
    If e.data.GetFormat(EXLISTLib.exClipboardFormatEnum.exCFFiles) Then
        If (e.data.Files.Count > 0) Then
            AxList1.BeginUpdate()
            With AxList1.Items
                Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum, iFrom As Integer =
AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
                Dim i As Long
                For i = 0 To e.data.Files.Count - 1
                    Dim newI As Integer = .Add(e.data.Files(i))
                    If (iFrom >= 0) Then
                        .ItemPosition(newI) = .ItemPosition(iFrom)
                    End If
                Next
            End With
            AxList1.EndUpdate()
        End If
    End If
End Sub
```

The following C# sample inserts a child item for each file that user drags:

```
private void axList1_OLEDragDrop(object sender,
AxEXLISTLib._IListEvents_OLEDragDropEvent e)
```

```
{
    if (e.data.GetFormat(Convert.ToInt16(EXLISTLib.exClipboardFormatEnum.exCFFiles)))
        if (e.data.Files.Count > 0)
        {
            EXLISTLib.HitTestInfoEnum hit;
            int c = 0, iFrom = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
            axList1.BeginUpdate();
            for (int i = 0; i < e.data.Files.Count; i++)
            {
                int newI = axList1.Items.Add(e.data.Files[i].ToString());
                if (iFrom >= 0)
                    axList1.Items.set_ItemPosition(newI, axList1.Items.get_ItemPosition(iFrom));
            }
            axList1.EndUpdate();
        }
}
```

The following VFP sample inserts a child item for each file that user drags:

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

local c, hit, iFrom
c = 0
hit = 0
if ( data.GetFormat( 15 ) ) && exCFFiles
    if ( data.Files.Count() > 0 )
        with thisform.List1.Items
            iFrom = thisform.List1.ItemFromPoint( x, y, @c, @hit )
            thisform.List1.BeginUpdate()
            for i = 0 to data.files.Count() - 1
                local newI
                newI = .Add( data.files(i) )
                if ( iFrom >= 0 )
                    .ItemPosition(newI) = .ItemPosition(iFrom)
                endif
            next
```

```
        thisform.List1.EndUpdate()
      endwith
    endif
endif
```

event OLEDragOver (Data as ExDataObject, Effect as Long, Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, State as Integer)

Occurs when one component is dragged over another.

Type	Description
Data as 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 Remarks.
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.

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 in Remarks.

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,
AxEXLISTLib._IListEvents_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,Exlistlib_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

VB6

```
Private Sub OLEDragOver(ByVal Data As EXLISTLibCtl.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(oList,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.List.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.List.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.List.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 in Remarks.
DefaultCursors as Boolean	Boolean value that determines whether to use the default mouse cursor, or to use a user-defined mouse cursor.True (default) = use default mouse cursor.False = do not use default cursor. Mouse cursor must be set with the MousePointer property of the Screen object.

The settings for Effect are:

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

If there is no code in the OLEGiveFeedback event, or if the defaultcursors parameter is set to True, the mouse cursor will be set to the default cursor provided by the control. The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:
If Effect = exOLEDropEffectCopy...
Instead, the source component should mask for the value or values being sought, such as this:
If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...
-or-
If (Effect And exOLEDropEffectCopy)...
This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

The control supports only manual OLE drag and drop events.

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

C#
```
// OLEGiveFeedback event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLEGiveFeedback event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

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

C#
```
private void OLEGiveFeedback(object sender,
AxEXLISTLib._IListEvents_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_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(oList,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, AxEXLISTLib._IListEvents_OLESetDataEvent e)
{
}
```

C++
```
void OnOLESetData(LPDISPATCH Data,short Format)
{
}
```

C++ Builder
```
void __fastcall OLESetData(TObject *Sender,Exlistlib_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:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_OLESetDataEvent) Handles OLESetData
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXLISTLibCtl.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(oList,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.List.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

In the /NET Assembly, you have to use the DragEnter event as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

Use the Background(exDragDropBefore) property to specify the visual appearance for the dragging items, before painting the items. Use the Background(exDragDropAfter) property to specify the visual appearance for the dragging items, after painting the items. Use the Background(exDragDropList) property to specify the graphic feedback for the item from the cursor, while the OLE drag and drop operation is running.

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.

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, AxEXLISTLib._IListEvents_OLEStartDragEvent e)
{
}
```

C++

```
void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
{
}
```

C++ Builder

```
void __fastcall OLEStartDrag(TObject *Sender,Exlistlib_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: AxEXLISTLib._IListEvents_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 AxEXLISTLib._IListEvents_OLEStartDragEvent) Handles OLEStartDrag
End Sub
```

VB6

```
Private Sub OLEStartDrag(ByVal Data As
EXLISTLibCtl.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(oList,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.List.1::IExDataObject,AllowedEffects as N)
end function
```

dBASE

```
function nativeObject_OLEStartDrag(Data,AllowedEffects)
```

```
return
```

The idea of drag and drop in exList control is the same as in other controls. To start accepting drag and drop sources the exList control should have the OLEDropMode to exOLEDropManual. Once that is is set, the exList starts accepting any drag and drop sources.

The first step is if you want to be able to drag items from your exList control to other controls the idea is to handle the OLE_StartDrag event. The event passes an object ExDataObject (Data) as argument. The Data and AllowedEffects should be set inside of handler event. Here's a sample:

```
Private Sub List1_OLEStartDrag(ByVal Data As EXLISTLibCtl.IExDataObject, AllowedEffects
As Long)
   Dim i As Long
   Dim str As String
   With List1.Items
      For i = 0 To .SelectCount - 1
         str = str + .Caption(.SelectedItem(i), 0)
         str = str + vbCrLf
      Next
      AllowedEffects = 1
      Data.SetData str, exCFText
   End With
End Sub
```

What the above code does? It takes each selected item from the exList source, and builds a string that will be passed to the clipboard object Data as text string. So, the clipboard data will contains text, so you will be able to drag the items to a text control. Of course the target controls should have enabled the OLE drag and drop features, and it depends on each control. The AllowedEffects = 1 specifies the type of cursor used in drag and drop operations. The value should be a combination of one of the exOLEDropEffectEnum type. Please check your environment browsed for the all possible values: 1 - Copy, 2 - Move, 0 - None, So, AllowedEffects = 1 specifies the "copy" cursor. If you don't set the AllowedEffects parameter, the cursor is by default None. There is no rule how you have to store the data into the clipboard Data object. It depends how you would like to do the drag and drop operations works.

The second step is accepting OLE drag and drop source objects. That means, if you would like to let your control accept drag and drop objects, you have to handle the OLEDragDrop

event. It passes as argument an object Data that stores the drag and drop information. The next sample shows how to take each line saved into the data object and add for each line found a new item into your exList control:

```
Private Sub List2_OLEDragDrop(ByVal Data As Object, Effect As Long, ByVal Button As
Integer, ByVal Shift As Integer, ByVal X As Long, ByVal Y As Long)
  Dim str As String
  str = Data.GetData(exCFText)
  if Not (Right(str, Len(vbCrLf)) = vbCrLf) Then
    str = str + vbCrLf
  End If
  With List2
    .BeginUpdate
    Dim c As Long, i As Long, n1 As Long, n2 As Long, nPos As Long
    While .Items.SelectCount <> 0
      .Items.SelectItem(.Items.SelectedItem(0)) = False
    Wend
    i = .ItemFromPoint(X / 15, Y / 15, c)
    If (i >= 0) Then
      nPos = .Items.ItemPosition(i)
    End If
    Debug.Print "Index = " & i
    n1 = 1
    n2 = InStr(n1, str, vbCrLf)
    While (n2 > 0)
      Dim n As Long
      n = .Items.Add(Mid(str, n1, n2 - n1))
      If .Items.Caption(n, 0) = "" Then
        .Items.ItemBreak(n) = DotLine
      End If
      If (i >= 0) Then
        .Items.ItemPosition(n) = nPos + 1
        nPos = nPos + 1
      End If
      .Items.SelectItem(n) = True
      n1 = n2 + Len(vbCrLf)
      n2 = InStr(n1, str, vbCrLf)
```

```
        Wend
        .EndUpdate
    End With
End Sub
```

The following VC sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
#import <exlist.dll> rename( "GetItems", "exGetItems" )

#include "Items.h"
#include "Columns.h"

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

void OnOLEStartDragList1(LPDISPATCH Data, long FAR* AllowedEffects)
{
    CItems items = m_list.GetItems();
    long nCount = items.GetSelectCount(), nColumnCount =
m_list.GetColumns().GetCount();
    if ( nCount > 0 )
    {
        *AllowedEffects = /*exOLEDropEffectCopy */ 1;
        EXLISTLib::IExDataObjectPtr spData( Data );
        if ( spData !=NULL )
        {
```

```
        CString strData;
        for ( long i = 0; i < nCount; i++ )
        {
            long nItem = items.GetSelectedItem( i );
            for ( long j = 0; j < nColumnCount; j++ )
                strData += V2S( &items.GetCaption( nItem, COleVariant( j ) ) ) + "\t";
        }
        strData += "\r\n";
        spData->SetData( COleVariant( strData ), COleVariant( (long)EXLISTLib::exCFText) );
    }
  }
}
```

The sample saves data as CF_TEXT format (EXLISTLib::exCFText). The data is a text, where each item is separated by "\r\n" (new line), and each cell is separated by "\t" (TAB charcater). Of course, data can be saved as you want. The sample only gives an idea of what and how it could be done. The sample uses the #import statement to import the control's type library, including definitions for ExDataObject and ExDataObjectFiles that are required to fill data to be dragged. If your exlist.dll file is located in another place than your system folder, the path to the exlist.dll file needs to be specified, else compiler errors occur.

The following VB.NET sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
Private Sub AxList1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_OLEStartDragEvent) Handles AxList1.OLEStartDrag
  With AxList1.Items
    If (.SelectCount > 0) Then
      e.allowedEffects = 1 'exOLEDropEffectCopy
      Dim i As Integer, j As Integer, strData As String, nColumnCount As Long =
AxList1.Columns.Count
      For i = 0 To .SelectCount - 1
        For j = 0 To nColumnCount - 1
          strData = strData + .Caption(.SelectedItem(i), j) + Chr(Keys.Tab)
        Next
      Next
      strData = strData + vbCrLf
      e.data.SetData(strData, EXLISTLib.exClipboardFormatEnum.exCFText)
    End If
```

```
    End With
End Sub
```

The following C# sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
private void axList1_OLEStartDrag(object sender,
AxEXLISTLib._IListEvents_OLEStartDragEvent e)
{
    int nCount = axList1.Items.SelectCount;
    if ( nCount > 0 )
    {
        int nColumnCount = axList1.Columns.Count;
        e.allowedEffects = /*exOLEDropEffectCopy*/ 1;
        string strData = "";
        for ( int i =0 ; i < nCount; i++ )
        {
            for ( int j = 0; j < nColumnCount; j++ )
            {
                object strCell = axList1.Items.get_Caption(axList1.Items.get_SelectedItem(i), j);
                strData +=  ( strCell != null ? strCell.ToString() : "" ) + "\t";
            }
            strData += "\r\n";
        }
        e.data.SetData( strData, EXLISTLib.exClipboardFormatEnum.exCFText );
    }
}
```

The following VFP sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
*** ActiveX Control Event ***
LPARAMETERS data, allowedeffects

local sData, nColumnCount, i, j
with thisform.List1.Items
    if ( .SelectCount() > 0 )
        allowedeffects = 1 && exOLEDropEffectCopy
```

```
    sData = ""
    nColumnCount = thisform.List1.Columns.Count
    for i = 0 to .SelectCount - 1
      for j = 0 to nColumnCount
        sData = sData + .Caption( .SelectedItem(i), j ) + chr(9)
      next
      sData = sData + chr(10)+ chr(13)
    next
    data.SetData( sData, 1 ) && exCFText
  endif
endwith
```

event OversizeChanged (Horizontal as Boolean, NewVal as Long)

Occurs when the right range of the scroll is changed.

Type	Description
Horizontal as Boolean	A boolean expression that indicates whether the horizontal scroll bar is changed. If the Horizontal parameter is True the range of the horizontal scroll bar is changed. If the Horizontal parameter is False, the range of the vertical scroll bar is changed.
NewVal as Long	A long value that indicates the new scroll bar value.

If the control has no scroll bars the OffsetChanged and OversizeChanged events are not fired. When the scroll bar range is changed the OversizeChanged event is fired. Use the ScrollBars property of the control to determine which scroll bars are visible within the control. The control fires the LayoutChanged event when the user resizes a column, or change its position.

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

C#

```
private void OversizeChanged(object sender,bool Horizontal,int NewVal)
{
}
```

VB

```
Private Sub OversizeChanged(ByVal sender As System.Object,ByVal Horizontal As Boolean,ByVal NewVal As Integer) Handles OversizeChanged
End Sub
```

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

C#

```
private void OversizeChanged(object sender,
AxEXLISTLib._IListEvents_OversizeChangedEvent e)
{
}
```

C++

```
void OnOversizeChanged(BOOL Horizontal,long NewVal)
{
}
```

C++ Builder

```
void __fastcall OversizeChanged(TObject *Sender,VARIANT_BOOL Horizontal,long NewVal)
```

```
{
}
```

Delphi

```
procedure OversizeChanged(ASender: TObject; Horizontal : WordBool;NewVal :
Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OversizeChanged(sender: System.Object; e:
AxEXLISTLib._IListEvents_OversizeChangedEvent);
begin
end;
```

Powe…

```
begin event OversizeChanged(boolean Horizontal,long NewVal)
end event OversizeChanged
```

VB.NET

```
Private Sub OversizeChanged(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_OversizeChangedEvent) Handles OversizeChanged
End Sub
```

VB6

```
Private Sub OversizeChanged(ByVal Horizontal As Boolean,ByVal NewVal As Long)
End Sub
```

VBA

```
Private Sub OversizeChanged(ByVal Horizontal As Boolean,ByVal NewVal As Long)
End Sub
```

VFP

```
LPARAMETERS Horizontal,NewVal
```

Xbas…

```
PROCEDURE OnOversizeChanged(oList,Horizontal,NewVal)
RETURN
```

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

Java…

```
<SCRIPT EVENT="OversizeChanged(Horizontal,NewVal)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

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

```
Function OversizeChanged(Horizontal,NewVal)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOversizeChanged Boolean llHorizontal Integer llNewVal
    Forward Send OnComOversizeChanged llHorizontal llNewVal
End_Procedure
```

Visual Objects

```
METHOD OCX_OversizeChanged(Horizontal,NewVal) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OversizeChanged(boolean _Horizontal,int _NewVal)
{
}
```

XBasic

```
function OversizeChanged as v (Horizontal as L,NewVal as N)
end function
```

dBASE

```
function nativeObject_OversizeChanged(Horizontal,NewVal)
return
```

event RClick ()

Fired when right mouse button is clicked

Type	Description

Use the RClick event to add your context menu. The RClick event notifies your application when the user right clicks the control. Use the Click event to notify your application that the user clicks the control (using the left mouse button). Use the MouseDown or MouseUp event if you require the cursor position during the RClick event. Use the RClickSelect property to specify whether the user can select items by right clicking the mouse. Use the ItemFromPoint property to get the item from point. Use the ColumnFromPoint property to get the column from point.

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

C#

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

VB

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

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

C#

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

C++

```
void OnRClick()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event RClick()
end event RClick
```

VB.NET

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

VB6

```
Private Sub RClick()
End Sub
```

VBA

```
Private Sub RClick()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnRClick(oList)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComRClick
   Forward Send OnComRClick
```

```
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_RClick()
{
}
```

XBasic

```
function RClick as v ()
end function
```

dBASE

```
function nativeObject_RClick()
return
```

The following VB sample displays the cell over the cursor:

```
Private Sub List1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim c As Long, i As Long, hit As HitTestInfoEnum
   With List1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
      If (i >= 0) Then
         If (c >= 0) Then
            Debug.Print .Items.Caption(i, c)
         End If
      End If
   End With
End Sub
```

The following C++ sample displays the cell over the cursor:

```
void OnMouseUpList1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, i = m_list.GetItemFromPoint( X, Y, &c, &hit );
   if ( i >= 0 )
   {
      CItems items = m_list.GetItems();
      CString strCaption = V2S( &items.GetCaption( i, COleVariant( c ) ) );
```

```
        OutputDebugString( strCaption );
    }
}
```

The following VB.NET sample displays the cell over the cursor:

```
Private Sub AxList1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXLISTLib._IListEvents_MouseUpEvent) Handles AxList1.MouseUpEvent
    Dim c As Integer, hit As EXLISTLib.HitTestInfoEnum
    Dim i As Integer = AxList1.get_ItemFromPoint(e.x, e.y, c, hit)
    If (i >= 0) Then
        With AxList1.Items
            Debug.Write(.Caption(i, c))
        End With
    End If
End Sub
```

The following C# sample displays the cell over the cursor:

```
private void axList1_MouseUpEvent(object sender,
AxEXLISTLib._IListEvents_MouseUpEvent e)
{
    EXLISTLib.HitTestInfoEnum hit;
    int c = 0, i = axList1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i >= 0)
    {
        System.Diagnostics.Debug.WriteLine(axList1.Items.get_Caption(i, c).ToString());
    }
}
```

The following VFP sample displays the cell over the cursor:

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

local c, i, hit
With thisform.List1
    c = 0
    hit = 0
```

```
    i = .ItemFromPoint(x, y, @c, @hit)
    If (i >= 0)
        wait window nowait .Items.Caption(i, c)
    EndIf
EndWith
```

event RemoveColumn (Column as Column)

Fired before deleting a Column.

Type	Description
Column as Column	A Column object being removed from the Columns collection.

The RemoveColumn event is invoked when the control is about to remove a column. Use the RemoveColumn event to release any extra data associated to the column. Use the Remove method to remove a specific column from Columns collection. Use the Clear method to clear the columns collection. Use the Remove method to remove an item. Use the RemoveAll method to remove all items. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item. Use the Data property to assign an extra data to a column.

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

C#

```
private void RemoveColumn(object sender,exontrol.EXLISTLib.Column Column)
{
}
```

VB

```
Private Sub RemoveColumn(ByVal sender As System.Object,ByVal Column As
exontrol.EXLISTLib.Column) Handles RemoveColumn
End Sub
```

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

C#

```
private void RemoveColumn(object sender,
AxEXLISTLib._IListEvents_RemoveColumnEvent e)
{
}
```

C++

```
void OnRemoveColumn(LPDISPATCH Column)
{
}
```

C++ Builder

```
void __fastcall RemoveColumn(TObject *Sender,Exlistlib_tlb::IColumn *Column)
{
}
```

Delphi

```
procedure RemoveColumn(ASender: TObject; Column : IColumn);
begin
end;
```

Delphi 8 (.NET only)

```
procedure RemoveColumn(sender: System.Object; e:
AxEXLISTLib._IListEvents_RemoveColumnEvent);
begin
end;
```

Powe…

```
begin event RemoveColumn(oleobject Column)
end event RemoveColumn
```

VB.NET

```
Private Sub RemoveColumn(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_RemoveColumnEvent) Handles RemoveColumn
End Sub
```

VB6

```
Private Sub RemoveColumn(ByVal Column As EXLISTLibCtl.IColumn)
End Sub
```

VBA

```
Private Sub RemoveColumn(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnRemoveColumn(oList,Column)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComRemoveColumn Variant llColumn
   Forward Send OnComRemoveColumn llColumn
End_Procedure
```

Visual Objects

```
METHOD OCX_RemoveColumn(Column) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_RemoveColumn(COM _Column)
{
}
```

XBasic

```
function RemoveColumn as v (Column as OLE::Exontrol.List.1::IColumn)
end function
```

dBASE

```
function nativeObject_RemoveColumn(Column)
return
```

event RemoveItem (ItemIndex as Long)

Occurs before deleting an Item.

Type	Description
ItemIndex as Long	A long expression that indicates the index of the item being removed.

Use the RemoveItem to release any extra data that you might have used. The control fires the RemoveItem event before removing the item. Use the Remove method to remove an item from Items collection. Use the RemoveAll method to clear the items collection. Use the Remove method to remove a column. Use the Clear method to clear the columns collection. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item. Use the Data property to assign an extra data to a column.

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

C#

```
private void RemoveItem(object sender,int ItemIndex)
{
}
```

VB

```
Private Sub RemoveItem(ByVal sender As System.Object,ByVal ItemIndex As
Integer) Handles RemoveItem
End Sub
```

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

C#

```
private void RemoveItem(object sender,
AxEXLISTLib._IListEvents_RemoveItemEvent e)
{
}
```

C++

```
void OnRemoveItem(long ItemIndex)
{
}
```

C++ Builder

```
void __fastcall RemoveItem(TObject *Sender,long ItemIndex)
{
}
```

Delphi

```
procedure RemoveItem(ASender: TObject; ItemIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure RemoveItem(sender: System.Object; e:
AxEXLISTLib._IListEvents_RemoveItemEvent);
begin
end;
```

Powe…

```
begin event RemoveItem(long ItemIndex)
end event RemoveItem
```

VB.NET

```
Private Sub RemoveItem(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_RemoveItemEvent) Handles RemoveItem
End Sub
```

VB6

```
Private Sub RemoveItem(ByVal ItemIndex As Long)
End Sub
```

VBA

```
Private Sub RemoveItem(ByVal ItemIndex As Long)
End Sub
```

VFP

```
LPARAMETERS ItemIndex
```

Xbas…

```
PROCEDURE OnRemoveItem(oList,ItemIndex)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComRemoveItem Integer llItemIndex
   Forward Send OnComRemoveItem llItemIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_RemoveItem(ItemIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_RemoveItem(int _ItemIndex)
{
}
```

XBasic

```
function RemoveItem as v (ItemIndex as N)
end function
```

dBASE

```
function nativeObject_RemoveItem(ItemIndex)
return
```

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 scrollbar 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 OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. 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.EXLISTLib.ScrollBarEnum
ScrollBar,exontrol.EXLISTLib.ScrollPartEnum ScrollPart)
{
}
```

VB

```
Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As
exontrol.EXLISTLib.ScrollBarEnum,ByVal ScrollPart As
exontrol.EXLISTLib.ScrollPartEnum) Handles ScrollButtonClick
End Sub
```

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

C#

```
private void ScrollButtonClick(object sender,
AxEXLISTLib._IListEvents_ScrollButtonClickEvent e)
{
}
```

C++

```
void OnScrollButtonClick(long ScrollBar,long ScrollPart)
{
}
```

C++ Builder

```
void __fastcall ScrollButtonClick(TObject *Sender,Exlistlib_tlb::ScrollBarEnum
ScrollBar,Exlistlib_tlb::ScrollPartEnum ScrollPart)
{
}
```

Delphi

```
procedure ScrollButtonClick(ASender: TObject; ScrollBar :
ScrollBarEnum;ScrollPart : ScrollPartEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ScrollButtonClick(sender: System.Object; e:
AxEXLISTLib._IListEvents_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
AxEXLISTLib._IListEvents_ScrollButtonClickEvent) Handles ScrollButtonClick
End Sub
```

VB6

```
Private Sub ScrollButtonClick(ByVal ScrollBar As EXLISTLibCtl.ScrollBarEnum,ByVal
ScrollPart As EXLISTLibCtl.ScrollPartEnum)
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart
```

Xbas…

```
PROCEDURE OnScrollButtonClick(oList,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.List.1::ScrollBarEnum,ScrollPart as
OLE::Exontrol.List.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 List1
   .BeginUpdate
      .ScrollBars = exDisableBoth
      .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
      .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
```

```
    .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
  .EndUpdate
End With
```

```
Private Sub List1_ScrollButtonClick(ByVal ScrollPart As EXLISTLibCtl.ScrollPartEnum)
  MsgBox (ScrollPart)
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxList1
  .BeginUpdate()
  .ScrollBars = EXLISTLib.ScrollBarsEnum.exDisableBoth
  .set_ScrollPartVisible(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part Or EXLISTLib.ScrollPartEnum.exRightB1Part, True)
  .set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
  .set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exRightB1Part, "<img></img>2")
  .EndUpdate()
End With
```

```
Private Sub AxList1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_ScrollButtonClickEvent) Handles AxList1.ScrollButtonClick
    MessageBox.Show( e.scrollPart.ToString())
End Sub
```

The following C# sample displays the identifier of the scroll's button being clicked:

```
axList1.BeginUpdate();
axList1.ScrollBars = EXLISTLib.ScrollBarsEnum.exDisableBoth;
axList1.set_ScrollPartVisible(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part | EXLISTLib.ScrollPartEnum.exRightB1Part, true);
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axList1.set_ScrollPartCaption(EXLISTLib.ScrollBarEnum.exVScroll,
EXLISTLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axList1.EndUpdate();
```

```
private void axList1_ScrollButtonClick(object sender,
AxEXLISTLib._IListEvents_ScrollButtonClickEvent e)
{
   MessageBox.Show(e.scrollPart.ToString());
}
```

The following C++ sample displays the identifier of the scroll's button being clicked:

```
m_list.BeginUpdate();
m_list.SetScrollBars( 15 /*exDisableBoth*/ );
m_list.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32 /*exRightB1Part*/,
TRUE );
m_list.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_list.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2") );
m_list.EndUpdate();
```

```
void OnScrollButtonClickList1(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.List1
   .BeginUpdate
      .ScrollBars = 15
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWit
```

event SelectionChanged ()

Fired after a new item is selected.

Type	Description

Use the SelectionChanged event to notify your application that the user selects an item (that's selectable). The control supports single or multiple selection as well. When an item is selected or unselected the control fires the SelectionChanged event. Use the SingleSel property to specify if your control supports single or multiple selection. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. Use the SelForeColor and SelBackColor properties to specify colors for selected items.

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(oList)
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 items:

```
Private Sub List1_SelectionChanged()
On Error Resume Next
   Dim i As Long, j As Long, nCols As Long, nSels As Long, h As Long
   nCols = List1.Columns.Count
   With List1.Items
   nSels = .SelectCount
   For i = 0 To nSels - 1
      Dim s As String
      For j = 0 To nCols - 1
         s = s + .Caption(.SelectedItem(i), j) + Chr(9)
      Next
   Debug.Print s
   Next
   End With
End Sub
```

The following VB sample displays the selected items:

```
Private Sub List1_SelectionChanged()
```

```
    Dim i As Long
    With List1.Items
        For i = 0 To .SelectCount - 1
            Debug.Print .Caption(.SelectedItem(i), 0)
        Next
    End With
End Sub
```

The following C++ sample displays the selected items:

```
#include "Items.h"

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

void OnSelectionChangedList1()
{
    CItems items = m_list.GetItems();
    for ( long i = 0; i < items.GetSelectCount(); i++ )
    {
        long nItem = items.GetSelectedItem( i );
        CString strOutput;
        strOutput.Format( "%s\n", V2S( &items.GetCaption( nItem, COleVariant( (long)0 ) ) ) );
        OutputDebugString( strOutput );
    }
}
```

The following VB.NET sample displays the selected items:

```
Private Sub AxList1_SelectionChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxList1.SelectionChanged
   With AxList1.Items
      Dim i As Integer
      For i = 0 To .SelectCount - 1
         Debug.WriteLine(.Caption(.SelectedItem(i), 0))
      Next
   End With
End Sub
```

The following C# sample displays the selected items:

```
private void axList1_SelectionChanged(object sender, System.EventArgs e)
{
   for ( int i = 0; i < axList1.Items.SelectCount - 1; i++ )
   {
      object cell = axList1.Items.get_Caption( axList1.Items.get_SelectedItem( i), 0 );
      System.Diagnostics.Debug.WriteLine( cell != null ? cell.ToString() : "" );
   }
}
```

The following VFP sample displays the selected items:

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

with thisform.List1.Items
   for i = 0 to .SelectCount - 1
      wait window nowait .Caption( .SelectedItem( i ), 0 )
   next
endwith
```

event Sort ()

Fired when the control sorts a column.

Type	Description

The control fires the Sort event when the control sorts a column (the user clicks the column's head) or when the sorting position is changed in the control's sort bar. Use the SortOnClick property to specify the action that control executes when the user clicks the column's head. Use the SortBarVisible property to show the control's sort bar. Use the SortOrder property to sorts a column at runtime. Use the SortPosition property to determine the position of the column in the sorting columns collection. Use the ItemBySortPosition property to access a column giving its position in the sorting columns collection. Use the Sort event to sort the data when the SortOnClk property is exUserSort. Use the SingleSort property to allow sorting by single or multiple columns

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

C#

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

VB

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

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

C#

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

C++

```
void OnSort()
{
}
```

C++ Builder

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

Delphi

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

```
end;
```

Delphi 8 (.NET only)

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

Powe…

```
begin event Sort()
end event Sort
```

VB.NET

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

VB6

```
Private Sub Sort()
End Sub
```

VBA

```
Private Sub Sort()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnSort(oList)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComSort
   Forward Send OnComSort
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_Sort()
{
}
```

XBasic

```
function Sort as v ()
end function
```

dBASE

```
function nativeObject_Sort()
return
```

The following VB sample displays the list of columns being sorted:

```
Private Sub List1_Sort()
   Dim s As String, i As Long, c As Column
   i = 0
   With List1.Columns
      Set c = .ItemBySortPosition(i)
      While (Not c Is Nothing)
         s = s + """" & c.Caption & """ " & IIf(c.SortOrder = SortAscending, "A", "D") & " "
         i = i + 1
         Set c = .ItemBySortPosition(i)
      Wend
   End With
   s = "Sort: " & s
   Debug.Print s
End Sub
```

The following VC sample displays the list of columns being sorted:

```
void OnSortList1()
{
   CString strOutput;
   CColumns columns = m_list.GetColumns();
   long i = 0;
   CColumn column = columns.GetItemBySortPosition( COleVariant( i ) );
```

```
    while ( column.m_lpDispatch )
    {
        strOutput += "\"" + column.GetCaption() + "\" " + ( column.GetSortOrder() == 1 ?
"A" : "D" ) + " ";
        i++;
        column = columns.GetItemBySortPosition( COleVariant( i ) );
    }
    strOutput += "\r\n";
    OutputDebugString( strOutput );
}
```

The following C# sample displays the list of columns being sorted:

```
private void axList1_Sort(object sender, System.EventArgs e)
{
    string strOutput = "";
    int i = 0;
    EXLISTLib.Column column = axList1.Columns.get_ItemBySortPosition( i );
    while ( column != null )
    {
        strOutput += column.Caption + " " + ( column.SortOrder ==
EXLISTLib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
        column = axList1.Columns.get_ItemBySortPosition( ++i );
    }
    Debug.WriteLine( strOutput );
}
```

The following VB.NET sample displays the list of columns being sorted:

```
Private Sub AxList1_Sort(ByVal sender As Object, ByVal e As System.EventArgs) Handles
AxList1.Sort
    With AxList1
        Dim s As String, i As Integer, c As EXLISTLib.Column
        i = 0
        With AxList1.Columns
            c = .ItemBySortPosition(i)
            While (Not c Is Nothing)
                s = s + """" & c.Caption & """ " & IIf(c.SortOrder =
```

```
EXLISTLib.SortOrderEnum.SortAscending, "A", "D") & " "
            i = i + 1
            c = .ItemBySortPosition(i)
        End While
    End With
    s = "Sort: " & s
    Debug.WriteLine(s)
  End With
End Sub
```

The following VFP sample displays the list of columns being sorted:

```
local s, i, c
i = 0
s = ""
With thisform.List1.Columns
  c = .ItemBySortPosition(i)
  do While (!isnull(c))
    with c
      s = s + "'" + .Caption
      s = s + "' " + IIf(.SortOrder = 1, "A", "D") + " "
      i = i + 1
    endwith
    c = .ItemBySortPosition(i)
  enddo
endwith
s = "Sort: " + s
wait window nowait s
```

event ToolTip (ItemIndex as Long, ColIndex as Long, Visible as Boolean, X as Long, Y as Long, CX as Long, CY as Long)

Fired when the control prepares the object's tooltip.

Type	Description
ItemIndex as Long	A long expression that indicates the item's index or -1 if the cursor is not over the cell.
ColIndex as Long	A long expression that indicates the column's index.
Visible as Boolean	A boolean expression that indicates whether the object's tooltip is visible.
X as Long	A long expression that indicates the left location of the tooltip's window. The x values is always expressed in screen coordinates.
Y as Long	A long expression that indicates the top location of the tooltip's window. The y values is always expressed in screen coordinates.
CX as Long	A long expression that indicates the width of the tooltip's window.
CY as Long	A long expression that indicates the height of the tooltip's window.

The ToolTip event notifies your application that the control prepares the tooltip for a cell or column. Use the ToolTip event to change the default position of the tooltip's window. Use the CellToolTip property to specify the cell's tooltip. Use the Tooltip property to assign a tooltip to a column. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipDelay property to specify the time in ms that passes before the ToolTip appears. Use the ToolTipPopDelay property to specify the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

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

C#

```
private void ToolTip(object sender,int ItemIndex,int ColIndex,ref bool Visible,ref int X,ref int Y,int CX,int CY)
{
}
```

VB

```
Private Sub ToolTip(ByVal sender As System.Object,ByVal ItemIndex As Integer,ByVal ColIndex As Integer,ByRef Visible As Boolean,ByRef X As Integer,ByRef Y As Integer,ByVal CX As Integer,ByVal CY As Integer) Handles ToolTip
```

```
End Sub
```

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

C#

```
private void ToolTip(object sender, AxEXLISTLib._IListEvents_ToolTipEvent e)
{
}
```

C++

```
void OnToolTip(long ItemIndex,long ColIndex,BOOL FAR* Visible,long FAR* X,long
FAR* Y,long CX,long CY)
{
}
```

C++ Builder

```
void __fastcall ToolTip(TObject *Sender,long ItemIndex,long
ColIndex,VARIANT_BOOL * Visible,long * X,long * Y,long CX,long CY)
{
}
```

Delphi

```
procedure ToolTip(ASender: TObject; ItemIndex : Integer;ColIndex : Integer;var
Visible : WordBool;var X : Integer;var Y : Integer;CX : Integer;CY : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ToolTip(sender: System.Object; e:
AxEXLISTLib._IListEvents_ToolTipEvent);
begin
end;
```

Powe…

```
begin event ToolTip(long ItemIndex,long ColIndex,boolean Visible,long X,long
Y,long CX,long CY)
end event ToolTip
```

VB.NET

```
Private Sub ToolTip(ByVal sender As System.Object, ByVal e As
AxEXLISTLib._IListEvents_ToolTipEvent) Handles ToolTip
End Sub
```

VB6

```
Private Sub ToolTip(ByVal ItemIndex As Long,ByVal ColIndex As Long,Visible As
Boolean,X As Long,Y As Long,ByVal CX As Long,ByVal CY As Long)
```

```
End Sub
```

VBA

```
Private Sub ToolTip(ByVal ItemIndex As Long,ByVal ColIndex As Long,Visible As Boolean,X As Long,Y As Long,ByVal CX As Long,ByVal CY As Long)
End Sub
```

VFP

```
LPARAMETERS ItemIndex,ColIndex,Visible,X,Y,CX,CY
```

Xbas…

```
PROCEDURE OnToolTip(oList,ItemIndex,ColIndex,Visible,X,Y,CX,CY)
RETURN
```

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

Java…

```
<SCRIPT EVENT="ToolTip(ItemIndex,ColIndex,Visible,X,Y,CX,CY)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ToolTip(ItemIndex,ColIndex,Visible,X,Y,CX,CY)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComToolTip Integer llItemIndex Integer llColIndex Boolean llVisible Integer llX Integer llY Integer llCX Integer llCY
    Forward Send OnComToolTip llItemIndex llColIndex llVisible llX llY llCX llCY
End_Procedure
```

Visual Objects

```
METHOD OCX_ToolTip(ItemIndex,ColIndex,Visible,X,Y,CX,CY) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ToolTip(int _ItemIndex,int _ColIndex,COMVariant /*bool*/ _Visible,COMVariant /*long*/ _X,COMVariant /*long*/ _Y,int _CX,int _CY)
{
}
```

XBasic

```
function ToolTip as v (ItemIndex as N,ColIndex as N,Visible as L,X as N,Y as N,CX as N,CY as N)
```

```
end function
```

dBASE

```
function nativeObject_ToolTip(ItemIndex,ColIndex,Visible,X,Y,CX,CY)
return
```

Expressions

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

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

Usage examples:

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

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

The constants can be represented as:

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

0x00FF00, and so so.

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

The predefined constants are:

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

The supported binary arithmetic operators are:

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

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

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

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

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

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

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

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

  variable := expression

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

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

  =: variable

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

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

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

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

  expression ? true_part : false_part

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

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

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

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

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

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

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

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

statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

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

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

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

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

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

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

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

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using iif and or expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. The type operator may return any of the following: 0 - empty (not initialized), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, **7 - date**, **8 - string**, 9 - object, 10 - error, **11 - boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bites. For instance *type(%1) = 8* specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, hex(`FF`) returns 255, while the hex(255) or hex(0xFF) returns the `FF` string. The hex(hex(`FFFFFFFF`)) always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

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

2)

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

The operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as $1,000.00, for US format.
- value **format** 'flags' (binary operator) formats the value with specified flags. If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the *1000 format ''* displays 1,000.00 for English format, while 1.000,00 is displayed for German format. 1000 format '2|.|3|,' will always displays 1,000.00 no matter of settings in the control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

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

  - *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
  - *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
  - Grouping - indicates the number of digits in each group of numbers to the left of

the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.

- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
  - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
  - 1 - Negative sign, number; for example, -1.1
  - 2 - Negative sign, space, number; for example, - 1.1
  - 3 - Number, negative sign; for example, 1.1-
  - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

The operators for strings are:

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

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

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

The operators for dates are:

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

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

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