



ExComboBox

The ExComboBox control allows you to display your tabular or hierarchical data within a drop-down list. A combo box is a commonly used graphical user interface widget (or control) that's a combination of a drop-down list or list box and a single-line editable textbox, allowing the user to either type a value directly or select a value from the list. The ExComboBox component lets the user changes its visual appearance using skins, each one providing an additional visual experience that enhances viewing pleasure.

Features of the full version include:

- **Simple, DropDown** and **DropDownList** styles
- Multiple Selection Support
- Multiple Columns Support
- Clear-Button support
- **Print** and Print Preview support.
- **Skinnable Interface** support (ability to apply a skin to any background part)
- Images, check boxes, radio buttons support.
- 'start with' and 'contains ' **incremental searching** support
- Mouse wheel support
- **FilterBar** Support. Ability to filter items with an easy-to-use interface
- **FilterFor** Support. Ability to filter items on the fly with an always-visible edit control
- Ability to filter items using patterns that include **wild card characters** like *, ? or #, items between two **dates** or **numbers**, with an easy-to-use interface
- Ability to filter items using OR, AND or NOT operators between columns
- **Conditional Format** support
- **Computed Fields** support
- Prompt support
- Owner Draw Support.
- Data bounding
- **Multiple edit controls**
- **Multiple levels header** support
- Sorting columns support
- **Locked columns** support
- **Locked/Fixed rows/items** support
- **"Merge Cells"** support
- Any cell supports **Built-in HTML format**
- Ability to assign HTML multi-lines tooltips to cells
- Ability to show the control's element from right-to-left for Hebrew, Arabic and other **RTL** languages
- Ability to assign multiple icons to a cell

- Ability to resize the drop-down window, using the size grip
- Ability to change the look for expanding buttons or hierarchy lines
- Ability to handle divider items, or merging cells
- Ability to specify **non selectable** items
- Unlimited color and font options for any item or cell
- Items with different height, multi-line items

Baciu, Cluj (5.1.1) ↓

S	Code	Country/State	Status	Function	Date	Coordinates
<input type="checkbox"/>	QKJ	Kenchela (3.15.1)	RL	3	0907	3527N 00708E
<input checked="" type="checkbox"/>	FR	France (4)				
<input checked="" type="checkbox"/>	RO	Romania (5)				
<input type="checkbox"/>	CJ	Cluj (5.1)				
<input type="checkbox"/>	BCU	Baciu, Cluj (5.1.1)	RL	2, 3	0407	4648N 02331E
<input type="checkbox"/>	DEJ	Dej (5.1.2)	RL	3	0407	4709N 02352E
<input type="checkbox"/>	RZG	Huedin (5.1.3)	RL	6	0901	4652N 02303E
<input type="checkbox"/>	RZK	Jucu-Herghelia (5.1.4)	RL	6	0901	4651N 02348E
<input type="checkbox"/>	MVU	Mihai Viteazu (5.1.5)	RL	3	0507	4632N 02345E
<input type="checkbox"/>	TDA	Turda (5.1.6)	RL	2, 3	1101	4634N 02347E
<input type="checkbox"/>	TR5	Tureni (5.1.7)	RL	1, 6	1407	4637N 02342E
<input type="checkbox"/>	AE	United Arab Emirates (6)				
<input checked="" type="checkbox"/>	GB	United Kingdom (7)				
<input type="checkbox"/>	ABE	Aberdeen City (7.1)				
<input type="checkbox"/>	BGF	Bridge of Don (7.1.1)	RL	3	0901	5711N 00206W
<input type="checkbox"/>	ZAH	Bucksburn (7.1.2)	RL	6	0901	5710N 00210W
<input type="checkbox"/>	KWS	Kingswells (7.1.3)	RL	3	1401	5709N 00212W

Start Filter... Status = 'RL' S 401 result(s)

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

How to get support?

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

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

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

Don't forget that you can contact our development team if you have ideas or requests for new components, by sending us an e-mail at support@exontrol.com (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 ExComboBox 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 an handle. 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(h), and the cell as CellProperty(h,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.[AddItem](#) to add a root item, whenever you need to display data as a list, or the Items.[InsertItem](#) to insert child items when you need to display your data as a tree. The [PutItems](#) method takes an array of data and fetches in the control. The [InsertItem](#) event notifies your application that a new items 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 [CellCaption](#) property to specify the captions for the rest of the columns. The AddItem or InsertItem 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 a cell. 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.](#)

Š 1999-2006 [Exontrol Inc, Software](#). All rights reserved.

constants AlignmentEnum

The column's [Alignment](#) property uses the AlignmentEnum enumeration to specify the column's alignment. The [Alignment](#) property of the control changes the drop down list's alignment.

Name	Value	Description
LeftAlignment	0	The text is left aligned.
CenterAlignment	1	The text is center aligned.
RightAlignment	2	The text is right aligned

constants AppearanceEnum

The AppearanceEnum enumeration is used to specify the type of 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 drag and drop operation starts once the user clicks and moves the cursor up or down.

- The flag that ends on ...**OnShortTouch** indicates the action the control does when the user short touches the screen
- The flag that ends on ...**OnRight** indicates the action the control does when the user right clicks the control.
- The flag that ends on ...**OnLongTouch** indicates the action the control does when the user long touches the screen

The AutoDragEnum type supports the following values:

Name	Value	Description
exAutoDragNone	0	AutoDrag is disabled.
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.
		The item can be dragged to any position or to any parent, while the dragging object keeps its

exAutoDragPositionKeepInden2

indentation. This option can be used to allow the user change the item's position at runtime by drag and drop. In the same time, the parent's item could be changed but keeping the item's indentation. 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 or parent of the dragging items is changed. 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.

exAutoDragPositionAny

3

The item can be dragged to any position or to any parent, with no restriction. The dragging items could be the focused item or a contiguously selection. The parent of the dragging items could change with no restrictions, based on the position of the dragging item. Click the selection and moves the cursor up or down, so the position or parent of the dragging items is changed. Click the selection and moves the cursor left or right, so the item's indentation is decreased or increased. 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.

Click here  to watch a movie on how exAutoDragCopyText works.

exAutoDragCopy

8

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 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.

exAutoDragPositionOnShortTo256

The object can be dragged from a position to

another, but not outside of its group.

exAutoDragPositionKeepIndentationOnShortTouch 502502 The object can be dragged to any position or to any parent, while the dragging object keeps its indentation.

exAutoDragPositionAnyOnShortTouch 767767 The object can be dragged to any position or to any parent, with no restriction.

exAutoDragCopyOnShortTouch 20482048 Drag and drop the selected objects to a target application, and paste them as image or text.

exAutoDragCopyTextOnShortTouch 23042304 Drag and drop the selected objects to a target application, and paste them as text only.

exAutoDragCopyImageOnShortTouch 25602560 Drag and drop the selected objects to a target application, and paste them as image only.

exAutoDragCopySnapShotOnShortTouch 28162816 Drag and drop a snap shot of the current component.

exAutoDragScrollOnShortTouch 40964096 The component is scrolled by clicking the object and dragging to a new position.

exAutoDragPositionOnRight 6553665536 The object can be dragged from a position to another, but not outside of its group.

exAutoDragPositionKeepIndentationOnRight 101792101792 The object can be dragged to any position or to any parent, while the dragging object keeps its indentation.

exAutoDragPositionAnyOnRight 196608196608 The object can be dragged to any position or to any parent, with no restriction.

exAutoDragCopyOnRight 524288524288 Drag and drop the selected objects to a target application, and paste them as image or text.

exAutoDragCopyTextOnRight 589824589824 Drag and drop the selected objects to a target application, and paste them as text only.

exAutoDragCopyImageOnRight 655360655360 Drag and drop the selected objects to a target application, and paste them as image only.

exAutoDragCopySnapShotOnRight 720896720896 Drag and drop a snap shot of the current component.

exAutoDragScrollOnRight 10485761048576 The component is scrolled by clicking the object and dragging to a new position.

exAutoDragPositionOnLongTouch 1677721616777216 The object can be dragged from a position to another, but not outside of its group.

The object can be dragged to any position or to any parent, while the dragging object keeps its

exAutoDragPositionKeepIndentation 83554161

exAutoDragPositionAnyOnLongTouch 50331648 The object can be dragged to any position or to any parent, with no restriction.

exAutoDragCopyOnLongTouch 134217728 Drag and drop the selected objects to a target application, and paste them as image or text.

exAutoDragCopyTextOnLongTouch 150994944 Drag and drop the selected objects to a target application, and paste them as text only.

exAutoDragCopyImageOnLongTouch 167772160 Drag and drop the selected objects to a target application, and paste them as image only.

exAutoDragCopySnapShotOnLongTouch 184549760 Drag and drop a snap shot of the current component.

exAutoDragScrollOnLongTouch 268435456 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

*/*not supported in the lite version*/* 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. Use the DisplayFilterButton property to specify whether the drop down filter bar button is visible or hidden.
exFooterFilterBarButton	1	Specifies the background color for the closing button in the filter bar. Use the ClearFilter method to remove the filter from the control.
exCellButtonUp	2	Specifies the background color for the cell's button, when it is up. Use the CellHasButton property to assign a button to a cell.
exCellButtonDown	3	Specifies the background color for the cell's button, when it is down. Use the CellHasButton property to assign a button to a cell.

exDropDownButtonUp	4	Specifies the visual appearance for the drop down button, when it is up.
exDropDownButtonDown	5	Specifies the visual appearance for the drop down button, when it is down.
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.
exForeColorFilter	27	Specifies the foreground color for the drop down filter window.
exCursorHoverColumn	32	Specifies the visual appearance for the column when the cursor hovers the column. By default, the exCursorHoverColumn property is zero, and it has no effect, so the visual appearance for the column is not changed when the cursor hovers the header.
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	Indicates the tooltip's background color.
exToolTipForeColor	66	Indicates the tooltip's foreground color.
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	Specifies the visual appearance for the position sign between columns, when the user changes the position of the column by drag and drop.
exTreeLinesColor	186	Specifies the color to show the tree-lines (connecting lines from the parent to the children)
exClearButtonUp	190	Specifies the visual appearance for the clear button, when it is up. The ShowClearButton property shows or hides the control's clear-button.
exClearButtonDown	191	Specifies the visual appearance for the clear button, when it is down. The ShowClearButton property shows or hides the control's clear-button.
exSizeGrip	3	Specifies the visual appearance for the control's size grip.
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.
exVSTThumb	260	The thumb part (exThumbPart) in normal state.
exVSTThumbP	261	The thumb part (exThumbPart) when it is pressed.
exVSTThumbD	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.
		All button parts (L1-L5, LButton, exThumbPart,

exSBtnD	326	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.

constants CaptionFormatEnum

The CaptionFormatEnum type defines how the cell's caption is painted. Use the [CellCaptionFormat](#) property to specify whether the cell supports built-in HTML format.

Name	Value	Description
exText	0	No HTML tags are painted.
		<p>The control uses built-in HTML tags to display the caption using HTML format. The control supports the following HTML tags:</p> <ul style="list-style-type: none">• ... displays the text in bold• <i> ... </i> displays the text in <i>italics</i>• <u> ... </u> <u>underlines</u> 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 <i>AnchorClick(AnchorID, Options)</i> event when the user clicks the anchor element. The <i>FormatAnchor</i> property customizes the visual effect for anchor elements. <p>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.</p> <ul style="list-style-type: none">◦ exp, stores the plain text to be shown once the user clicks the anchor, such as "<code><a ;exp=show lines></code>"◦ e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "<code><a ;e64=gA8ABmABnABjABvABshIAOQAEAA</code>

" that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAAC 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=rrgbb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrgbb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrgbb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ...

- `</solidline>` draws a black solid-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.
- `<dotline rrggbb> ... </dotline>` or `<dotline=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
 - `number[:width]` 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.
 - `key[:width]` 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 **&**; (&), **<**; (<), **>**; (>), **"**; (") and **&#number;**; (the character with specified code), For instance, the **€** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a **#character** and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **** to define a smaller or a larger font to be displayed. For instance: "Text with **<off 6>**subscript" displays the text such as: Text with subscript
The "Text with **<off -6>**superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the **rr/gg/bb** represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The **** HTML tag can be used to define the height of the font. Any of the **rrggb**, **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 FFFFFFFF;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:



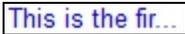
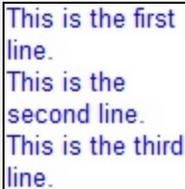
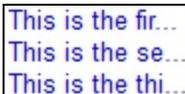
exComputedField

2

Indicates a computed field. The [CellCaption](#) 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	<p>Indicates that the cell's caption is displayed on a single line. In this case any <code>\r\n</code> or <code>
</code> HTML tags is ignored. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as:</p> 
exCaptionWordWrap	0	<p>Specifies that the cell's caption is displayed on multiple lines, by wrapping the words. Any <code>\r\n</code> or <code>
</code> HTML tag breaks the line. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as:</p> 
exCaptionBreakWrap	1	<p>Specifies that the cell's caption is displayed on multiple lines, by wrapping the breaks only. Only The <code>\r\n</code> or <code>
</code> HTML tag breaks the line. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as:</p> 

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. Use the PartialCheck property to allow partial check feature within the column. (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)
exCellButtonAutoWidth	3	Specifies that all buttons in the column fit the cell's caption. (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. (Color 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. (Color expression)

exCellVAlignment 6 Specifies the column's vertical alignment. By default, the Def(exCellVAlignment) property is exMiddle. Use the [CellVAlignment](#) property to specify the vertical alignment for a particular cell.

([VAlignmentEnum](#) expression, exMiddle)

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)

exCellCaptionFormat 17 Specifies that all cells in the column display HTML format, if it is [exHTML](#). Similar with the [CellCaptionFormat](#) property,

([CaptionFormatEnum](#) expression, exText)

Assigns an owner draw object for the entire column.

exCellOwnerDraw 33 Use the [CellOwnerDraw](#) property to assign an owner draw object to a single cell. (an object that implements the [IOwnerDrawHandler](#) interface)

exCellDrawPartsOrder 34 Specifies the order of the drawing parts for the entire column. By default, this option is "check,icon,icons,picture,caption", which means that 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")

exCellPaddingLeft 48 Gets or sets the left padding (space) of the cells within the column.

(*Long expression*)

exCellPaddingRight 49 Gets or sets the right padding (space) of the cells within the column.

(*Long expression*)

exCellPaddingTop 50 Gets or sets the top padding (space) of the cells within the column.

(*Long expression*)

exCellPaddingBottom 51 Gets or sets the bottom padding (space) of the cells within the column.

(*Long expression*)

exHeaderPaddingLeft 52 Gets or sets the left padding (space) of the column's header.

(*Long expression*)

exHeaderPaddingRight

53

Gets or sets the right padding (space) of the column's header.

(*Long expression*)

exHeaderPaddingTop

54

Gets or sets the top padding (space) of the column's header.

(*Long expression*)

exHeaderPaddingBottom

55

Gets or sets the bottom padding (space) of the column's header.

(*Long expression*)

exColumnResizeContiguously 64

exColumnResizeContiguously. Gets or sets a value that indicates whether the control's content is updated while the user is resizing the column.

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' text 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 delimited 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.
--------------------	----	--

exFilterForCaption	26	Defines the caption of 'Filter For' in the FilterFor field. Use the FilterForVisible property to display the control's FilterFor field.
--------------------	----	---

exFilterForTooltip	27	Defines the tooltip to be shown when cursor hovers the FilterFor field. Use the FilterForVisible property to display the control's FilterFor field.
--------------------	----	---

constants DividerAlignmentEnum

Defines the alignment for a divider line into a divider item. Use the [ItemDividerLineAlignment](#) property to align the line in a divider item. Use the [ItemDivider](#) property to add a divider item

Name	Value	Description
DividerBottom	0	The divider line is displayed on bottom side of the item.
DividerCenter	1	The divider line is displayed on center of the item.
DividerTop	2	The divider line is displayed at the top of the item.
DividerBoth	3	The divider line is displayed at the top and bottom of the item.

constants DividerLineEnum

Defines the type of divider line. The [ItemDividerLine](#) property uses the DividerLineEnum type.

Name	Value	Description
EmptyLine	0	No line
SingleLine	1	Single line
DoubleLine	2	Double line
DotLine	3	Dotted line
DoubleDotLine	4	Double Dotted line
ThinLine	5	Thin line
DoubleThinLine	6	Double thin line

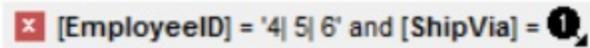
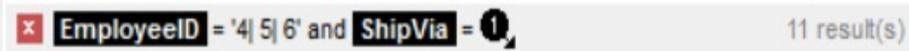
constants ExpandButtonEnum

Defines how the control displays the expanding/collapsing buttons.

Name	Value	Description
exNoButtons	0	The control displays no expand buttons.
exPlus	-1	A plus sign is displayed for collapsed items, and a minus sign for expanded items. (⊕ ⊖)
exArrow	1	The control uses icons to display the expand buttons. (▶ ▼)
exCircle	2	The control uses icons to display the expand buttons. (⊕ ⊙)
exWPlus	3	The control uses icons to display the expand buttons. (⊕ ⊖)
exCustom	4	The HasButtonsCustom property specifies the index of icons being used for +/- signs on parent items.

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

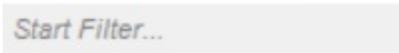
256

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 through 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

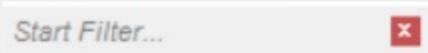
The exFilterBarShowCloseIfRequired flag indicates that the close button of the control's filter bar is displayed only if the control has any currently filter applied. The [Background\(exFooterFilterBarButton\)](#) property on -1 hides permanently the close button of the control's filter bar.



exFilterBarShowCloseOnRight

1024

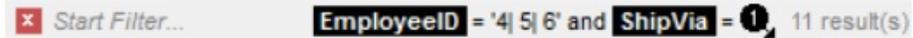
The exFilterBarShowCloseOnRight flag specifies that the close button of the control's filter bar should be displayed on the right side. If the control's [RightToLeft](#) property is True, the close button of the control's filter bar would be automatically displayed on the left side.



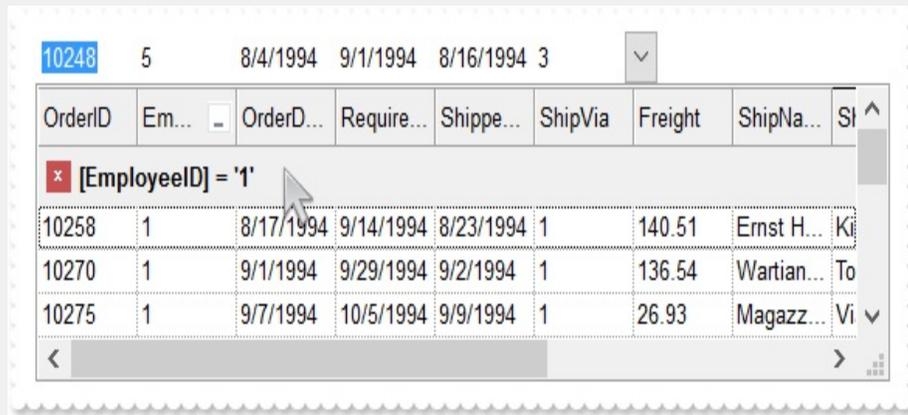
exFilterBarCompact

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.



The exFilterBarTop flag specifies that the control's filter bar is displayed on the top of the control (below the header and above the rows) as shown:

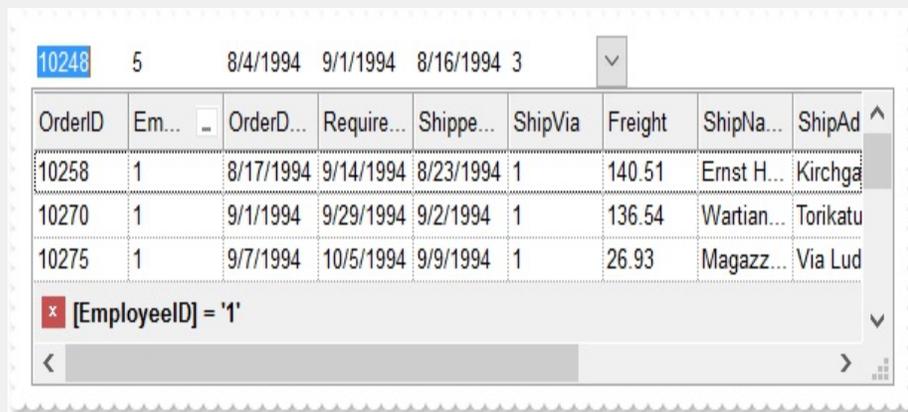


OrderID	Em...	OrderD...	Require...	Shippe...	ShipVia	Freight	ShipNa...	Si
10258	1	8/17/1994	9/14/1994	8/23/1994	1	140.51	Ernst H...	Ki
10270	1	9/1/1994	9/29/1994	9/2/1994	1	136.54	Wartian...	To
10275	1	9/7/1994	10/5/1994	9/9/1994	1	26.93	Magazz...	Vi

exFilterBarTop

8192

By default, the filter-bar is shown aligned to the bottom (between items and horizontal-scroll bar) as shown:



OrderID	Em...	OrderD...	Require...	Shippe...	ShipVia	Freight	ShipNa...	ShipAd
10258	1	8/17/1994	9/14/1994	8/23/1994	1	140.51	Ernst H...	Kirchga
10270	1	9/1/1994	9/29/1994	9/2/1994	1	136.54	Wartian...	Tonikatu
10275	1	9/7/1994	10/5/1994	9/9/1994	1	26.93	Magazz...	Via Lud

constants FilterIncludeEnum

The FilterIncludeEnum type defines the items to include when control's filter is applied. The [FilterInclude](#) property specifies the items being included, when the list is filtered. The FilterIncludeEnum type supports the following values:

Name	Value	Description
exItemsWithoutChilds	0	Items (and parent-items) that match the filter are shown (no child-items are included)
exItemsWithChilds	1	Items (parent and child-items) that match the filter are shown
exRootsWithoutChilds	2	Only root-items (excludes child-items) that match the filter are displayed
exRootsWithChilds	3	Root-items (and child-items) that match the filter are displayed
exMatchingItemsOnly	4	Shows only the items that matches the filter (no parent or child-items are included)
exMatchIncludeParent	240	Specifies that the item matches the filter if any of its parent-item matches the filter. The exMatchIncludeParent flag can be combined with any other value.

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 below:

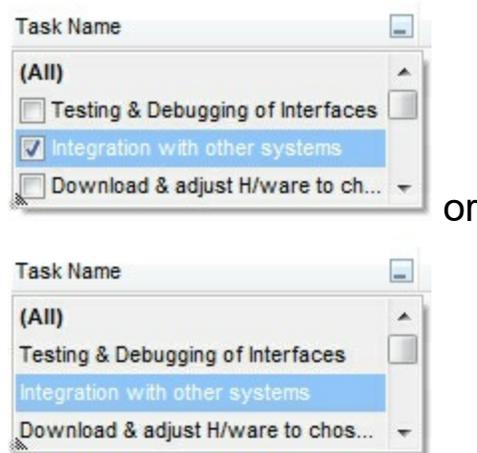
Name	Value	Description
exAllItems	0	The filter's list includes all items in the column.
exVisibleItems	1	The filter's list includes only visible (filtered) items from the column. The visible items include child items of collapsed items.
exNoItems	2	The filter's list does not include any item from the column. Use this option if the drop down filter displays a calendar control for instance.
exLeafItems	3	The filter's list includes the leaf items only. A leaf item is an item with no child items.
exRootItems	4	The filter's list includes the root items only.
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 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.

exShowCheckBox

256

The following screen shot shows the drop down filter **with** or **with no** exShowCheckBox flag:

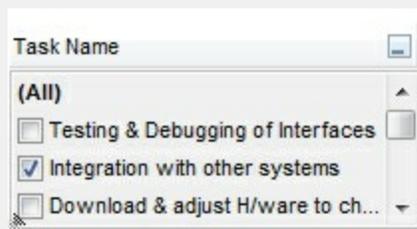


The selection background is not shown for checked items in the filter's list. This flag can be combined with exShowCheckBox.

exHideCheckSelect

512

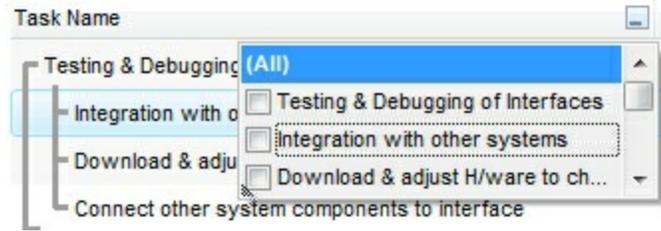
The following screen shot shows no selection background for the checked items:



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.

exShowFocusItem

1024 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

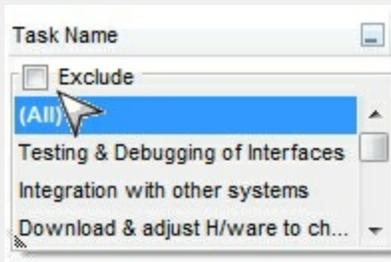
This flag indicates whether the filter's tooltip is shown. The [Description](#)(exFilterBarTooltip,exFilterBarPatternTool...) properties defines the filter's tooltips.

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: <ul style="list-style-type: none">• '?' 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

*/*not supported in the lite version*/* 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. By default, the filtering is case-insensitive.

The FilterTypeEnum supports the following values:

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 of the Column object 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. The ' ' character separates multiple patterns. If any of the *, ?, # or characters are preceded by a \ (escape character) it masks the character itself. This option can be combined with exFilterDoCaseSensitive flag to perform a case-sensitive filtering.
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.
		If the FilterType property is exNumeric, the Filter

exNumeric

5

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.

exCheck

6

Only checked or unchecked items are included. The [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. This option can be combined with exFilterDoCaseSensitive flag to perform a case-sensitive filtering. The [Filter](#) property indicates the value(s) to be filtered. The values are separated by '|' character.

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.

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

constants GridLineStyleEnum

The GridLineStyle 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 GridLineStyle 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 HierarchyLineEnum

Defines how the control paints the hierarchy lines.

Name	Value	Description
exNoLine	0	The control displays no lines when painting the hierarchy.
exDotLine	-1	The control uses a dotted line to paint the hierarchy.
exSolidLine	1	The control uses a solid line to paint the hierarchy.
exThinLine	2	The control uses a thin line to paint the hierarchy.

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.
exHTExpandButton	1	In the +/- button associated with a cell. The HasButtons property specifies whether the cell displays a +/- sign to let user expands the item.
exHTCellIndent	2	In the indentation associated with a cell. The Indent property retrieves or sets the amount, in pixels, that child items are indented relative to their parent items.
exHTCellInside	4	On the icon, picture, check or caption associated with a cell.
exHTCellCaption	20	(HEXA 14) In the caption associated with a cell. The CellCaption property specifies the cell's caption.
exHTCellCheck	36	(HEXA 24) In the check/radio button associated with a cell. The CellHasCheckBox or CellHasRadioButton property specifies whether the cell displays a checkbox or a radio button.
exHTCellIcon	68	(HEXA 44) In first icon associated with a cell. The CellImage or CellImages property specifies the cell's icon displayed next to the cell's caption.
exHTCellPicture	132	(HEXA 84). In a picture associated to a cell. Use the CellPicture property to assign a picture to a cell.
exHTCellCaptionIcon	1044	(HEXA 414) In the icon's area inside the cell's caption. The tag inserts an icon inside the cell's caption. The tag is valid only if the CellCaptionFormat property exHTML.

constants LinesAtRootEnum

Defines how the control displays the lines at root. The LinesAtRoot property defines the way the tree lines are shown. The HasLines property defines the type of the line to be shown. The HasButtons property defines the expand/collapse buttons for parent items.

The LinesAtRootEnum type support the following values:

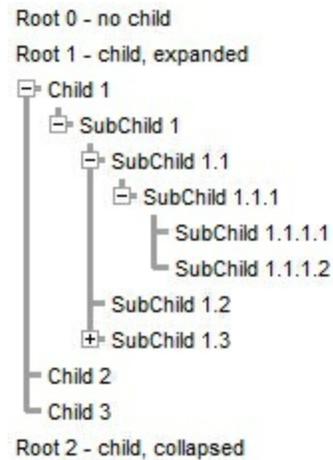
Name

Value Description

exNoLinesAtRoot

0

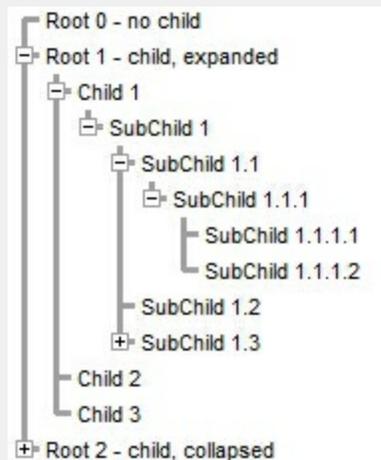
No lines at root items.



exLinesAtRoot

-1

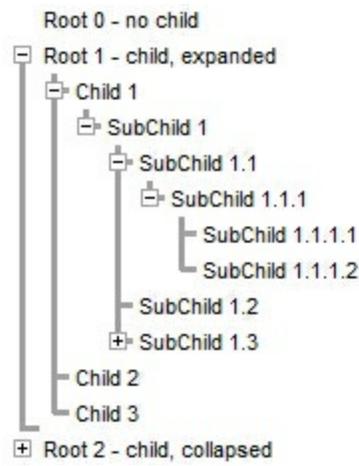
The control links the root items.



The control shows no links between roots, and divides them as being in the same group.

exGroupLinesAtRoot

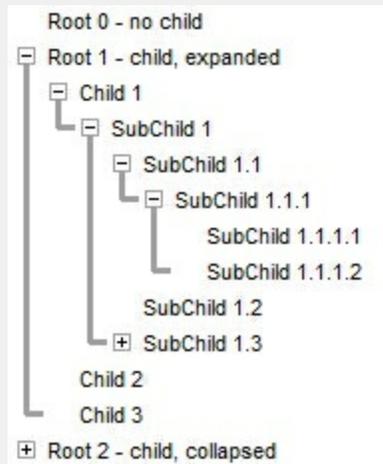
1



The lines between root items are no shown, and the links show the items being included in the group.

exGroupLines

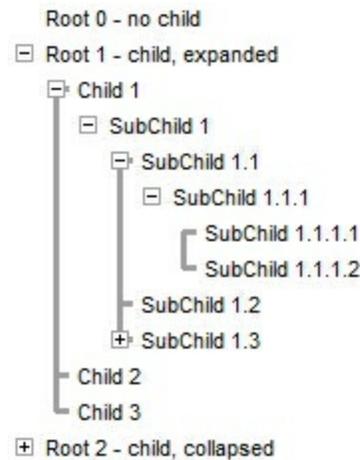
2



The lines between root items are no shown, and the links are shown between child only.

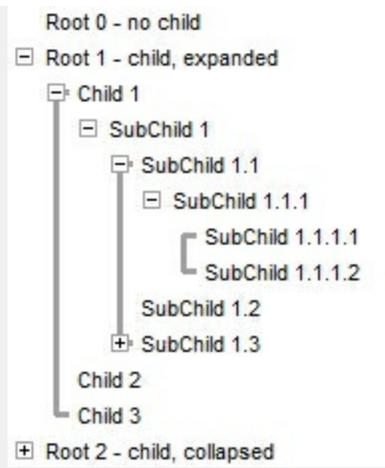
exGroupLinesInside

3



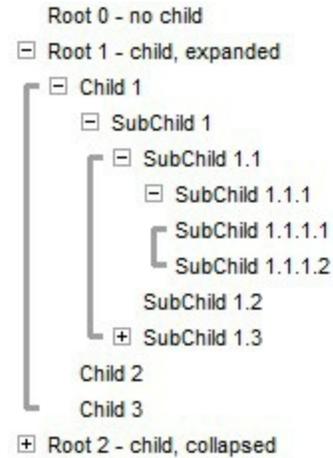
The lines between root items are no shown, and the links are shown for first and last visible child item.

exGroupLinesInsideLeaf 4



The lines between root items are no shown, and the links are shown for first and last visible child item. A parent item that contains flat child items only, does not indent the child part. By a flat child we mean an item that does not contain any child item.

exGroupLinesOutside 5



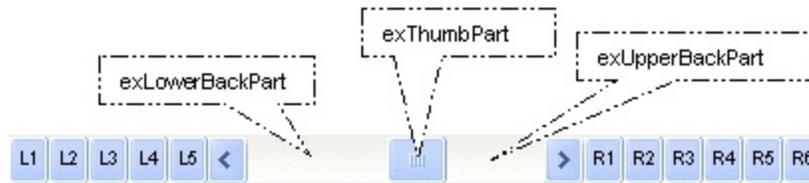
constants ScrollBarEnum

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

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

constants 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 fourth 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 the user clicks the column's header. Use the [SortOnClick](#) property to specify the action that control takes when user clicks on the column's header.

Name	Value	Description
exNoSort	0	The control does nothing when user clicks the column's header.
exDefaultSort	-1	The control sorts the column when user clicks the column's header.
exUserSort	1	The control displays the sort icon as it been sorted, but the column is not sorted.

constants SortOrderEnum

Specifies the column's sort order. Use the [SortOrder](#) property to change the column's sort order.

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

The SortTypeEnum enumeration defines the ways how the control can sort the columns. Use the [SortType](#) property to specify how the column gets sorted. The [CellCaption](#) property indicates the values being sorted.

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 CellData property indicates the values being sorted. Values are sorted as numbers.
SortUserDataString	6	The CellData property indicates the values being sorted. Values are sorted as strings.
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 StyleEnum

The StyleEnum enumeration defines the control's style. Use the [Style](#) property to specify the style of the control.

Name	Value	Description
Simple	0	The extended list(tree) control is displayed at all times. The current selection in the dropdown window is displayed in the edit control.
DropDown	1	Similar to Simple style, except that the dropdown window control is not displayed unless the user selects the drop down button next to the edit control.
DropDownList	2	Similar to DropDown style, except that the edit control is replaced by a static-text item that displays the current selection in the dropdown window.

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 UVisualThemeEnum

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

Name	Value	Description
exNoVisualTheme	0	exNoVisualTheme
exDefaultVisualTheme	16777215	exDefaultVisualTheme
exHeaderVisualTheme	1	exHeaderVisualTheme
exFilterBarVisualTheme	2	exFilterBarVisualTheme
exButtonsVisualTheme	4	exButtonsVisualTheme
exCalendarVisualTheme	8	exCalendarVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exBorderVisualTheme	1024	exBorderVisualTheme

constants **VAlignmentEnum**

Specifies the source's vertical alignment. Use the [CellVAlignment](#) property to align vertically the caption.

Name	Value	Description
exTop	0	exTop
exMiddle	1	exMiddle
exBottom	2	exBottom

Appearance object

*/*not supported in the lite version*/* 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)

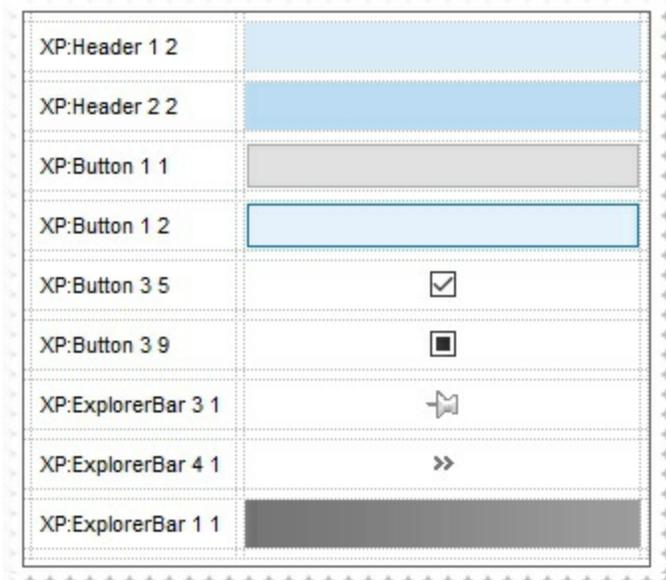
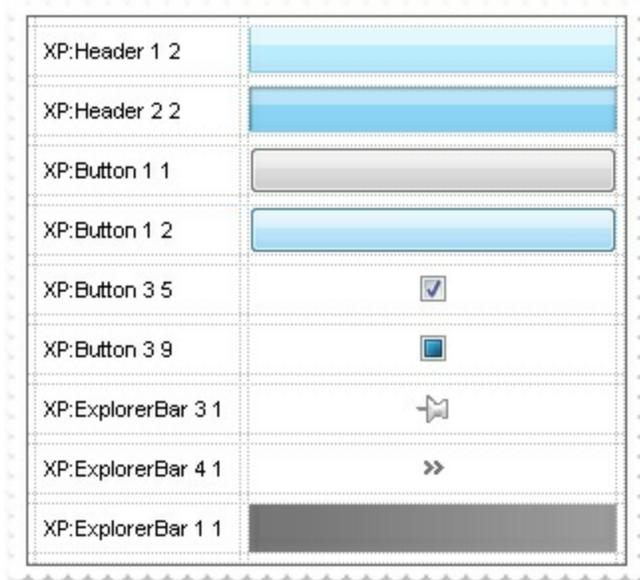
Adds or replaces a skin object to the control. */*not supported in the lite version*/*

Type	Description
ID as Long	<p>A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.</p> <p>The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the 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))</p> <p>If the Skin parameter points to a string expression, it can be one of the following:</p> <ul style="list-style-type: none">• 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.

Name	A	B	C	A+B+C
Root				
Child 1	7	3	1	11
Child 2	2	5	12	19
Child 3	14	1	4	16

The identifier you choose for the skin is very important to be used in the background properties like explained below. 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 screenshot shows a software interface with a tree view and a tooltip. The tree view has a root node 'Root 2' with four children: 'Child 1', 'Child 2', 'Child 3', and 'Not selectable item'. The tooltip for the 'EXONTROL' control lists features: multiple columns, tree view, incremental search, and built-in HTML support.

Name	Value
Child 1	2 - new item
Child 2	3 - new item
Child 3	4
Root 2	Not selectable item

EXONTROL's eXComboBox control.

Name
The Exontrol's eXComboBox control support several features including **multiple columns, tree view, incremental search, and built-in HTML support**, and so on.



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

- control's **borders** using the [Appearance](#) property
- visual effect of the borders of the drop down portion of the control, using the [DropDownBorder](#) property
- borders for the control's **tooltip**, using the [Background\(exToolTipAppearance\)](#) property
- control's **header bar**, [HeaderBackColor](#) 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
- **"drop down"** button, cell's **button**, **"drop down"** filter bar button, **"close"** filter bar button, **scrollbars**, and so on, [Background](#) property
- [CellImage](#), [CellImages](#), [HeaderImage](#), [CheckImage](#) or [RadiolImage](#), [HasButtonsCustom](#) property

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

```
With ComboBox1
    With .VisualAppearance
        .Add &H23, App.Path + "\selected.ebn"
    End With
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_combobox.GetVisualAppearance().Add( 0x23,  
COleVariant(_T("D:\\Temp\\ExComboBox_Help\\selected.ebn"))) );  
m_combobox.SetSelBackColor( 0x23000000 );  
m_combobox.SetSelForeColor( 0 );
```

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

```
With AxComboBox1  
  With .VisualAppearance  
    .Add(&H23, "D:\\Temp\\ExComboBox_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):

```
axComboBox1.VisualAppearance.Add(0x23,  
"D:\\Temp\\ExComboBox_Help\\selected.ebn");  
axComboBox1.Template = "SelBackColor = 587202560";
```

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

```
With thisform.ComboBox1  
  With .VisualAppearance  
    .Add(35, "D:\\Temp\\ExComboBox_Help\\selected.ebn")  
  EndWith
```

```
.SelForeColor = RGB(0, 0, 0)
```

```
.SelBackColor = 587202560
```

```
EndWith
```

The 587202560 value represents � in hexadecimal. The 32 value represents in hexadecimal

The first screen shot was generated using the following template:

```
BeginUpdate
```

```
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")
```

```
VisualAppearance.Add(5,"XP:ComboBox 1 1")
```

```
VisualAppearance.Add(6,"XP:ComboBox 1 3")
```

```
HeaderBackColor = 16777216
```

```
SelBackColor = 33554432
```

```
Background(1) = 50331648
```

```
Background(0) = 67108864
```

```
Background(20) = 33554432
```

```
Background(21) = 1
```

```
SelForeColor = 0
```

```
Background(4) = 83886080
```

```
Background(5) = 100663296
```

```
MarkSearchColumn = False
```

```
ColumnAutoResize = True
```

```
ShowFocusRect = False
```

```
ConditionalFormats
```

```
{
```

```
  Add("%1 >4")
```

```
  {
```

```
    Bold = True
```

```
    StrikeOut = True
```

```
    ForeColor = RGB(255,0,0)
```

```
    ApplyTo = 1
```

```
}  
Add("%2 > 4")  
{  
    Bold = True  
    StrikeOut = True  
    ForeColor = RGB(255,0,0)  
    ApplyTo = 2  
}  
Add("%3 > 4")  
{  
    Bold = True  
    StrikeOut = True  
    ForeColor = RGB(255,0,0)  
    ApplyTo = 3  
}  
Add("1")  
{  
    Bold = True  
    ApplyTo = 4  
}
```

```
}  
Columns  
{  
    "Name"  
    "A"  
    {  
        AllowSizing = False  
        Width = 24  
    }  
    "B"  
    {  
        AllowSizing = False  
        Width = 24  
    }  
    "C"  
    {  
        AllowSizing = False
```

```

Width = 24
}
"A+B+C"
{
    AllowSizing = False
    Width = 64
    ComputedField = "%1+%2+%3"
}
}
Items
{
    Dim h, h1
    h = AddItem("Root")
    CellCaptionFormat(h,4) = 1
    h1 = InsertItem(h,,"Child 1")
    CellCaption(h1,1) = 7
    CellCaption(h1,2) = 3
    CellCaption(h1,3) = 1
    h1 = InsertItem(h,,"Child 2")
    CellCaption(h1,1) = 2
    CellCaption(h1,2) = 5
    CellCaption(h1,3) = 12
    h1 = InsertItem(h,,"Child 3")
    CellCaption(h1,1) = 11
    CellCaption(h1,2) = 1
    CellCaption(h1,3) = 4
    ExpandItem(h) = True
}
EndUpdate

```

The second screen shot was generated using the following template:

```

BeginUpdate()
Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjIBAEijUlK8plUrlktl
Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjlgjIBAEijUlK8plUrlktl

```

Images("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0pIUrlkt

Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl

Images("gBJJgBAICAAJAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaGEaAIAEEbjMjIerlktl0

VisualAppearance

{

 ' Header

Add(1,"gBFLBCJwBAEHhEJAEGg4BawDg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAAGEYxR

 ' HeaderFilterBarButton

Add(2,"gBFLBCJwBAEHhEJAEGg4BAQEg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAAGEYxR

Add(3,"gBFLBCJwBAEHhEJAEGg4BBAEg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAAGEYxR

 ' SelectedItem

 Add(4,

"gBFLBCJwBAEHhEJAEGg4BV4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK

Add(7,"gBFLBCJwBAEHhEJAEGg4BAwEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

}

HeaderBackColor = 16777216 '0x01BBGGRR

BackColorSortBarCaption = 33488896 '0x01BBGGRR

FilterBarBackColor = 16777216 '0x01BBGGRR

Background(0) = 33554432 '0x02BBGGRR

Background(1) = 50331648 '0x03BBGGRR

Background(4) = 33554432 '0x02BBGGRR

Background(5) = 117440512 '0x07BBGGRR

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
BackColorSortBar = RGB(61,101,183)
FilterBarForeColor = RGB(255,255,255)
HeaderForeColor = RGB(255,255,255)
ForeColorSortBar = RGB(255,255,255)
SelForeColor = 0
ShowFocusRect = False
HeaderHeight = 20
SortBarHeight = 20

MarkSearchColumn = False
Alignment = 0
LinesAtRoot = 1
SortBarVisible = True
ColumnAutoResize = True
AllowSizeGrip = True
BackColor = RGB(255,255,255)
BackColorEdit = RGB(255,255,255)
BackColorLevelHeader = RGB(255,255,255)
FullRowSelect = False
DrawGridLines = -1
MinWidthList = 312
HeightList = 260
CheckImage(1) = 4
CheckImage(0) = 5
CheckImage(2) = 6
HasButtons = 4
HasButtonsCustom(0) = 7
HasButtonsCustom(1) = 8

Font

```
{  
  Name = "Trebuchet MS"  
}
```

Columns

```
{  
  "Name"  
  {  
    HeaderBold = True  
    DisplayFilterButton = True  
    LevelKey = 2  
    Width = 96  
  }  
  "Value"  
  {  
    LevelKey = 2  
    Width = 40  
  }  
  1  
  {  
    AllowSizing = False  
    HTMLCaption = "1 First"  
    Def(0) = True  
    LevelKey = 1  
    Width = 13  
  }  
  2  
  {  
    AllowSizing = False  
    HTMLCaption = "2 Second"  
    Def(0) = True  
    LevelKey = 1  
    Width = 13  
  }  
  3  
  {
```

AllowSizing = False

HTMLCaption = "3 Third"

Def(0) = True

LevelKey = 1

Width = 13

}

""

{

LevelKey = 1

Width = 20

}

}

AssignEditImageOnSelect(0) = True

Items

{

Dim h, h1, hx

h = AddItem("Root 1 (merged)")

CellImage(h,0) = 1

CellPicture(h,0) =

"gBHJJGHA5MIqAAXAD3AENhozhpmhqZhrMhr/h0QGcQM0QTMQZkQf8QAESGcSM0STM

ItemHeight(h) = 26

CellMerge(h,0) = 1

h1 = InsertItem(h, "Child 1")

CellImage(h1,1) = 3

CellHasCheckBox(h1,0) = True

CellCaption(h1,1) = "2 - new item"

CellCaptionFormat(h1,1) = 1

h1 = InsertItem(h, "Child 2")

CellImage(h1,1) = 3

CellCaption(h1,1) = "3 - new item"

CellHasCheckBox(h1,0) = True

CellState(h1,0) = 1

CellCaptionFormat(h1,1) = 1

```
ExpandItem(h) = True
```

```
h = AddItem("Root 2")
```

```
CellImage(h,0) = 1
```

```
ItemBold(h) = True
```

```
CellCaption(h,1) = "4"
```

```
h1 = InsertItem(h,,"Child 1")
```

```
CellImage(h1,0) = 1
```

```
h = AddItem()
```

```
CellCaption(h,1) = "Non selectable item"
```

```
CellToolTip(h,1) = "The user can't select non selectable items."
```

```
CellCaptionFormat(h,1) = 1
```

```
SelectableItem( h ) = False
```

```
ItemHeight(h) = 24
```

```
ItemDivider(h) = 1
```

```
ItemDividerLine(h) = 2
```

```
CellImage(h,1) = 3
```

```
CellHAlignment(h,1) = 1
```

```
EnableItem(h) = False
```

```
h = AddItem("Selectable...")
```

```
}
```

```
EndUpdate()
```

On **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
		CBS_UNCHECKED
		= 3
		CBS_UNCHECKED

BUTTON	BP_CHECKBOX = 3	= 4 CBS_CHECKEDH 5 CBS_CHECKEDPR CBS_CHECKEDDIS CBS_MIXEDNORM CBS_MIXEDHOT = CBS_MIXEDPRES CBS_MIXEDDISAB
	BP_GROUPBOX = 4	GBS_NORMAL = 1 GBS_DISABLED =
	BP_PUSHBUTTON = 1	PBS_NORMAL = 1 = 2 PBS_PRESSED PBS_DISABLED = PBS_DEFAULTED :
	BP_RADIOBUTTON = 2	RBS_UNCHECKED 1 RBS_UNCHECKE RBS_UNCHECKED = 3 RBS_UNCHECKED = 4 RBS_CHECKEDH 5 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 = ETS_READONLY = ETS_ASSIST = 7
EXPLORERBAR	EBP_HEADERBACKGROUND = 1	
	EBP_HEADERCLOSE = 2	EBHC_NORMAL = EBHC_HOT = 2 EBHC_PRESSED = EBHP_NORMAL =

EBP_HEADERPIN = 3

EBP_IEBARMENU = 4

EBP_NORMALGROUPBACKGROUND = 5

EBP_NORMALGROUPCOLLAPSE = 6

EBP_NORMALGROUPEXPAND = 7

EBP_NORMALGROUPHEAD = 8

EBP_SPECIALGROUPBACKGROUND = 9

EBP_SPECIALGROUPCOLLAPSE = 10

EBP_SPECIALGROUPEXPAND = 11

EBP_SPECIALGROUPHEAD = 12

HEADER

HP_HEADERITEM = 1

HP_HEADERITEMLEFT = 2

HP_HEADERITEMRIGHT = 3

HP_HEADERSORTARROW = 4

LISTVIEW

LVP_EMPTYTEXT = 5

LVP_LISTDETAIL = 3

LVP_LISTGROUP = 2

LVP_LISTITEM = 1

EBHP_HOT = 2
EBHP_PRESSED =
EBHP_SELECTED
4 EBHP_SELECTED
EBHP_SELECTED
6

EBM_NORMAL = 1
= 2 EBM_PRESSE

EBNGC_NORMAL :
EBNGC_HOT = 2
EBNGC_PRESSED
EBNGE_NORMAL :
EBNGE_HOT = 2
EBNGE_PRESSED

EBSGC_NORMAL :
EBSGC_HOT = 2
EBSGC_PRESSED
EBSGE_NORMAL :
EBSGE_HOT = 2
EBSGE_PRESSED

HIS_NORMAL = 1
2 HIS_PRESSED =
HILS_NORMAL = 1
= 2 HILS_PRESSE
HIRS_NORMAL = 1
= 2 HIRS_PRESSE
HSAS_SORTEDUP
HSAS_SORTEDDC

LIS_NORMAL = 1
2 LIS_SELECTED :
LIS_DISABLED = 4
LIS_SELECTEDNO
5

LVP_LISTSORTEDDETAIL = 4

MENU

MP_MENUBARDROPDOWN = 4

MP_MENUBARITEM = 3

MP_CHEVRON = 5

MP_MENUDROPDOWN = 2

MP_MENUITEM = 1

MP_SEPARATOR = 6

MENUBAND

MDP_NEWAPPBUTTON = 1

MDP_SEPERATOR = 2

PAGE

PGRP_DOWN = 2

PGRP_DOWNHORZ = 4

PGRP_UP = 1

PGRP_UPHORZ = 3

PROGRESS

PP_BAR = 1

PP_BARVERT = 2

MS_NORMAL = 1

MS_SELECTED = 2

MS_DEMOTED = 3

MS_NORMAL = 1

MS_SELECTED = 2

MS_DEMOTED = 3

MS_NORMAL = 1

MS_SELECTED = 2

MS_DEMOTED = 3

MS_NORMAL = 1

MS_SELECTED = 2

MS_DEMOTED = 3

MS_NORMAL = 1

MS_SELECTED = 2

MS_DEMOTED = 3

MS_NORMAL = 1

MS_SELECTED = 2

MS_DEMOTED = 3

MDS_NORMAL = 1

= 2 MDS_PRESSE

MDS_DISABLED =

MDS_CHECKED =

MDS_HOTCHECKE

DNS_NORMAL = 1

= 2 DNS_PRESSE

DNS_DISABLED =

DNHZS_NORMAL =

DNHZS_HOT = 2

DNHZS_PRESSED

DNHZS_DISABLED

UPS_NORMAL = 1

= 2 UPS_PRESSE

UPS_DISABLED =

UPHZS_NORMAL =

UPHZS_HOT = 2

UPHZS_PRESSED

UPHZS_DISABLED

PP_CHUNK = 3

REBAR

PP_CHUNKVERT = 4

RP_BAND = 3

RP_CHEVRON = 4

RP_CHEVRONVERT = 5

RP_GRIPPER = 1

RP_GRIPPERVERT = 2

CHEVS_NORMAL =
CHEVS_HOT = 2
CHEVS_PRESSED

SCROLLBAR

SBP_ARROWBTN = 1

SBP_GRIPPERHORZ = 8

SBP_GRIPPERVERT = 9

SBP_LOWERTRACKHORZ = 4

SBP_LOWERTRACKVERT = 6

SBP_THUMBBTNHORZ = 2

ABS_DOWNDISAB
ABS_DOWNHOT,
ABS_DOWNNORM
ABS_DOWNPRESS
ABS_UPDISABLED
ABS_UPHOT,
ABS_UPNORMAL,
ABS_UPPRESSED,
ABS_LEFTDISABLI
ABS_LEFTHOT,
ABS_LEFTNORMA
ABS_LEFTPRESSE
ABS_RIGHTDISAB
ABS_RIGHTHOT,
ABS_RIGHTNORM
ABS_RIGHTPRESS

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED
SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED
SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED
SCRBS_NORMAL :

SBP_THUMBBTNVERT = 3

SBP_UPPERTRACKHORZ = 5

SBP_UPPERTRACKVERT = 7

SBP_SIZEBOX = 10

SPIN

SPNP_DOWN = 2

SPNP_DOWNHORZ = 4

SPNP_UP = 1

SPNP_UPHORZ = 3

STARTPANEL

SPP_LOGOFF = 8

SPP_LOGOFFBUTTONS = 9

SPP_MOREPROGRAMS = 2

SPP_MOREPROGRAMSARROW = 3

SPP_PLACESLIST = 6

SPP_PLACESLISTSEPARATOR = 7

SPP_PREVIEW = 11

SPP_PROGLIST = 4

SPP_PROGLISTSEPARATOR = 5

SPP_USERPANE = 1

SCRBS_HOT = 2

SCRBS_PRESSED

SCRBS_DISABLED

SCRBS_NORMAL :

SCRBS_HOT = 2

SCRBS_PRESSED

SCRBS_DISABLED

SCRBS_NORMAL :

SCRBS_HOT = 2

SCRBS_PRESSED

SCRBS_DISABLED

SZB_RIGHTALIGN

SZB_LEFTALIGN =

DNS_NORMAL = 1

= 2 DNS_PRESSEI

DNS_DISABLED =

DNHZS_NORMAL =

DNHZS_HOT = 2

DNHZS_PRESSED

DNHZS_DISABLED

UPS_NORMAL = 1

= 2 UPS_PRESSEI

UPS_DISABLED =

UPHZS_NORMAL =

UPHZS_HOT = 2

UPHZS_PRESSED

UPHZS_DISABLED

SPLS_NORMAL =

SPLS_HOT = 2

SPLS_PRESSED =

SPS_NORMAL = 1

= 2 SPS_PRESSEI

STATUS

SPP_USERPICTURE = 10

SP_GRIPPER = 3

SP_PANE = 1

TAB

SP_GRIPPERPANE = 2

TABP_BODY = 10

TABP_PANE = 9

TABP_TABITEM = 1

TABP_TABITEMBOTHEDGE = 4

TABP_TABITEMLEFTEDGE = 2

TABP_TABITEMRIGHTEDGE = 3

TABP_TOPTABITEM = 5

TABP_TOPTABITEMBOTHEDGE = 8

TABP_TOPTABITEMLEFTEDGE = 6

TIS_NORMAL = 1
2 TIS_SELECTED :
TIS_DISABLED = 4
TIS_FOCUSED = 5
TIBES_NORMAL =
TIBES_HOT = 2
TIBES_SELECTED
TIBES_DISABLED
TIBES_FOCUSED :
TILES_NORMAL =
TILES_HOT = 2
TILES_SELECTED
TILES_DISABLED :
TILES_FOCUSED :
TIRES_NORMAL =
TIRES_HOT = 2
TIRES_SELECTED
TIRES_DISABLED
TIRES_FOCUSED :
TTIS_NORMAL = 1
= 2 TTIS_SELECTE
TTIS_DISABLED =
TTIS_FOCUSED =
TTIBES_NORMAL :
TTIBES_HOT = 2
TTIBES_SELECTEI
TTIBES_DISABLED
TTIBES_FOCUSED
TTILES_NORMAL :
TTILES_HOT = 2
TTILES_SELECTEI
TTILES_DISABLED
TTILES_FOCUSED
TTIRES_NORMAL

TABP_TOPTABITEMRIGHTEDGE = 7

TTIRES_HOT = 2
TTIRES_SELECTE
TTIRES_DISABLED
TTIRES_FOCUSEC

TASKBAND

TDP_GROUPCOUNT = 1
TDP_FLASHBUTTON = 2
TDP_FLASHBUTTONGROUPMENU = 3

TASKBAR

TBP_BACKGROUNDBOTTOM = 1
TBP_BACKGROUNDLEFT = 4
TBP_BACKGROUNDRIGHT = 2
TBP_BACKGROUNDTOP = 3
TBP_SIZINGBARBOTTOM = 5
TBP_SIZINGBARBOTTOMLEFT = 8
TBP_SIZINGBARRIGHT = 6
TBP_SIZINGBARTOP = 7

TOOLBAR

TP_BUTTON = 1

TP_DROPDOWNBUTTON = 2

TP_SPLITBUTTON = 3

TP_SPLITBUTTONDROPDOWN = 4

TP_SEPARATOR = 5

TS_NORMAL = 1 T
TS_PRESSED = 3
TS_DISABLED = 4
TS_CHECKED = 5
TS_HOTCHECKED
TS_NORMAL = 1 T
TS_PRESSED = 3
TS_DISABLED = 4
TS_CHECKED = 5
TS_HOTCHECKED
TS_NORMAL = 1 T
TS_PRESSED = 3
TS_DISABLED = 4
TS_CHECKED = 5
TS_HOTCHECKED
TS_NORMAL = 1 T
TS_PRESSED = 3
TS_DISABLED = 4
TS_CHECKED = 5
TS_HOTCHECKED
TS_NORMAL = 1 T
TS_PRESSED = 3
TS_DISABLED = 4
TS_CHECKED = 5
TS_HOTCHECKED

TP_SEPARATORVERT = 6

TOOLTIP

TTP_BALLOON = 3

TTP_BALLOONTITLE = 4

TTP_CLOSE = 5

TTP_STANDARD = 1

TTP_STANDARDTITLE = 2

TRACKBAR

TKP_THUMB = 3

TKP_THUMBBOTTOM = 4

TKP_THUMBLEFT = 7

TKP_THUMBRIGHT = 8

TKP_THUMBTOP = 5

TS_NORMAL = 1 T

TS_PRESSED = 3

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TTBS_NORMAL =

TTBS_LINK = 2

TTBS_NORMAL =

TTBS_LINK = 2

TTCS_NORMAL =

TTCS_HOT = 2

TTCS_PRESSED =

TTSS_NORMAL =

TTSS_LINK = 2

TTSS_NORMAL =

TTSS_LINK = 2

TUS_NORMAL = 1

2 TUS_PRESSED =

TUS_FOCUSED =

TUS_DISABLED =

TUBS_NORMAL =

TUBS_HOT = 2

TUBS_PRESSED =

TUBS_FOCUSED =

TUBS_DISABLED =

TUVLS_NORMAL =

TUVLS_HOT = 2

TUVLS_PRESSED

TUVLS_FOCUSED

TUVLS_DISABLED

TUVRS_NORMAL =

TUVRS_HOT = 2

TUVRS_PRESSED

TUVRS_FOCUSED

TUVRS_DISABLED

TUTS_NORMAL =

TUTS_HOT = 2

TUTS_PRESSED =

TUTS_FOCUSED =

TUTS_DISABLED =

TUVS_NORMAL =

TKP_THUMBVERT = 6

TKP_TICS = 9

TKP_TICSVERT = 10

TKP_TRACK = 1

TKP_TRACKVERT = 2

TRAYNOTIFY

TNP_ANIMBACKGROUND = 2

TNP_BACKGROUND = 1

TREEVIEW

TVP_BRANCH = 3

TVP_GLYPH = 2

TVP_TREEITEM = 1

WINDOW

WP_CAPTION = 1

WP_CAPTIONSIIZINGTEMPLATE = 30

WP_CLOSEBUTTON = 18

WP_DIALOG = 29

WP_FRAMEBOTTOM = 9

WP_FRAMEBOTTOMSIIZINGTEMPLATE = 36

WP_FRAMELEFT = 7

WP_FRAMELEFTSIIZINGTEMPLATE = 32

WP_FRAMERIGHT = 8

WP_FRAMERIGHTSIIZINGTEMPLATE = 34

WP_HELPBUTTON = 23

TUVS_HOT = 2

TUVS_PRESSED =

TUVS_FOCUSED =

TUVS_DISABLED =

TSS_NORMAL = 1

TSVS_NORMAL =

TRS_NORMAL = 1

TRVS_NORMAL =

GLPS_CLOSED =

GLPS_OPENED =

TREIS_NORMAL =

TREIS_HOT = 2

TREIS_SELECTED

TREIS_DISABLED

TREIS_SELECTED

= 5

CS_ACTIVE = 1 CS

= 2 CS_DISABLED

CBS_NORMAL = 1

= 2 CBS_PUSHED

CBS_DISABLED =

FS_ACTIVE = 1 FS

= 2

FS_ACTIVE = 1 FS

= 2

FS_ACTIVE = 1 FS

= 2

HBS_NORMAL = 1

= 2 HBS_PUSHED

HBS_DISABLED =

HSS_NORMAL = 1

WP_HORIZSCROLL = 25

= 2 HSS_PUSHED

HSS_DISABLED =

HTS_NORMAL = 1

WP_HORIZTHUMB = 26

2 HTS_PUSHED =

HTS_DISABLED =

MAXBS_NORMAL =

WP_MAX_BUTTON

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

RBS_DISABLED =

WP_SMALLCAPTION = 2

CS_ACTIVE = 1 CS

= 2 CS_DISABLED

WP_SMALLCAPTIONSIZINGTEMPLATE = 31

CBS_NORMAL = 1
= 2 CBS_PUSHED
CBS_DISABLED =
FS_ACTIVE = 1 FS
= 2

WP_SMALLCLOSEBUTTON = 19

WP_SMALLFRAMEBOTTOM = 12

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 =
MAXBS_NORMAL =
MAXBS_HOT = 2
MAXBS_PUSHED =
MAXBS_DISABLED

WP_SMALLMAXBUTTON

MXCS_ACTIVE = 1
MXCS_INACTIVE =
MXCS_DISABLED

WP_SMALLMAXCAPTION = 6

MNCS_ACTIVE = 1
MNCS_INACTIVE =
MNCS_DISABLED

WP_SMALLMINCAPTION = 4

RBS_NORMAL = 1
= 2 RBS_PUSHED
RBS_DISABLED =

WP_SMALLRESTOREBUTTON

SBS_NORMAL = 1
= 2 SBS_PUSHED
SBS_DISABLED =

WP_SMALLSYSBUTTON

SBS_NORMAL = 1
= 2 SBS_PUSHED
SBS_DISABLED =

WP_SYSBUTTON = 13

VSS_NORMAL = 1
= 2 VSS_PUSHED

WP_VERTSCROLL = 27

WP_VERTTHUMB = 28

VSS_DISABLED =
VTS_NORMAL = 1
2 VTS_PUSHED =
VTS_DISABLED =

method Appearance.Clear ()

Removes all skins in the control. */*not supported in the lite version*/*

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

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

method Appearance.Remove (ID as Long)

Removes a specific skin from the control. */*not supported in the lite version*/*

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 **header bar**, [HeaderBackColor](#) 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
- **"drop down"** button, cell's **button**, **"drop down"** filter bar button, **"close"** filter bar button, and so on, [Background](#) 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

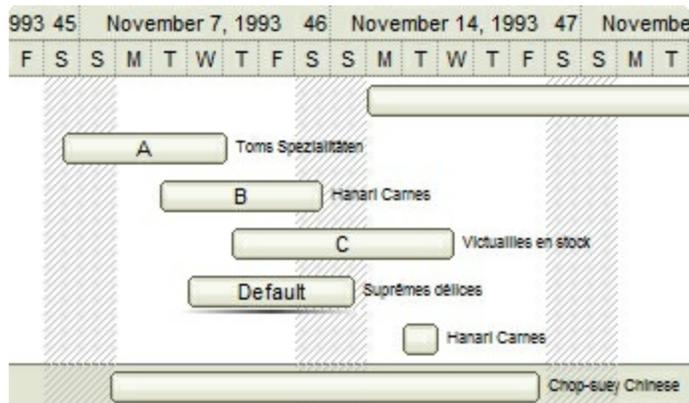
In the following screen shot the following objects displays the current EBN with a different color:

- "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

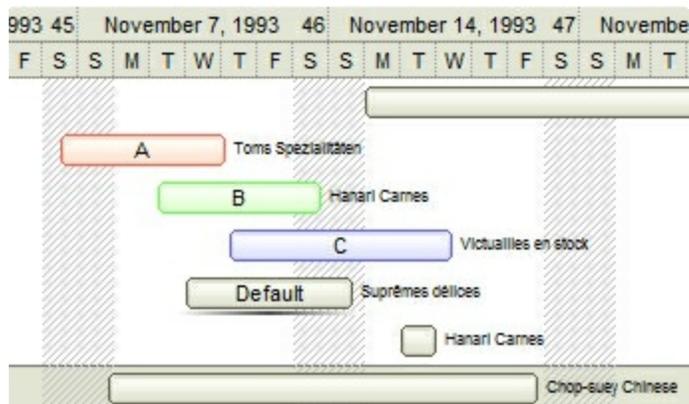
- "C" in Blue (RGB(0,0,255) , for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

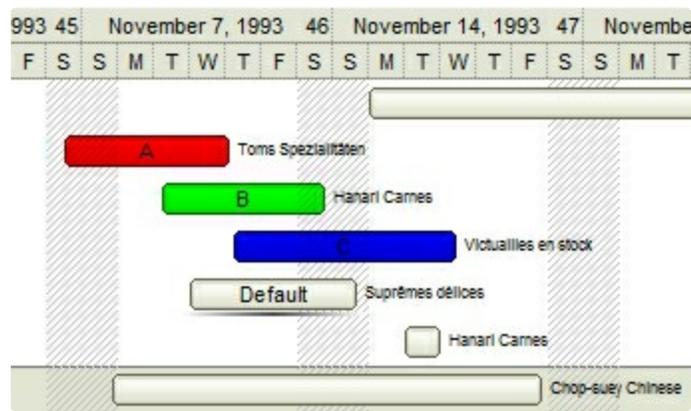
- **-3, no color is applied.** For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.



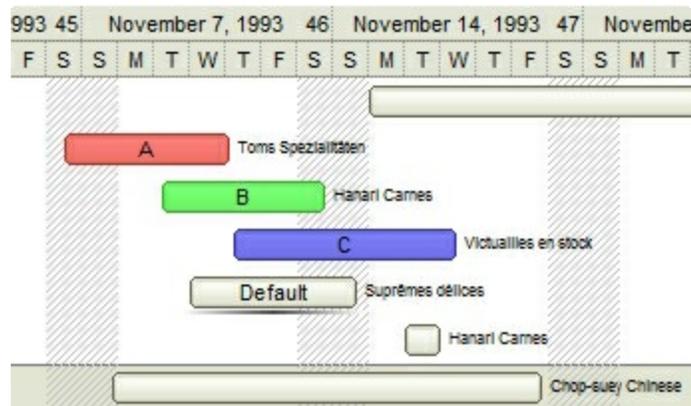
- **-2, OR-color scheme.** The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the 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), ...)

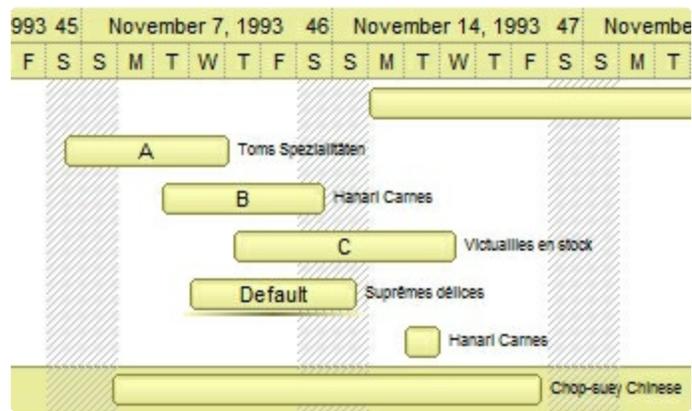


- **0, default**, the specified color is applied to the EBN. For instance, the `BackColorHeader = &H1FF0000`, applies a Blue color to the object. This option could be used to specify any color for the part of the components, that support EBN objects, not only solid colors.

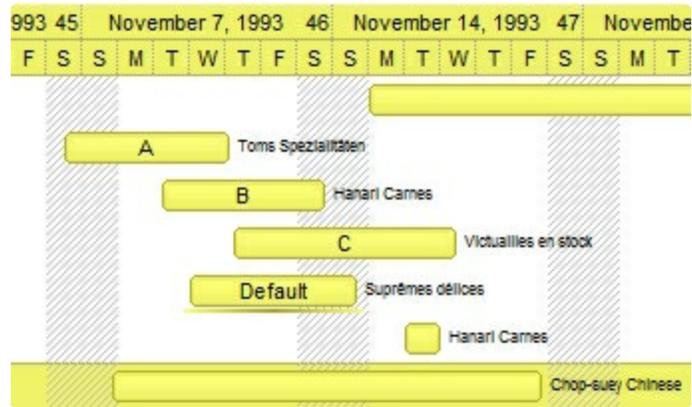


- **0xAABBGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the `RenderType` on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

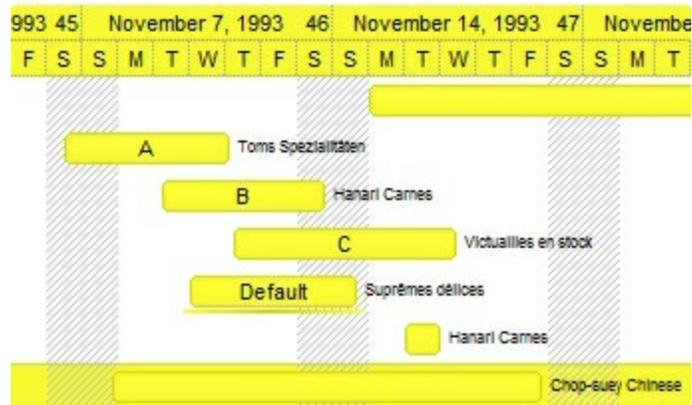
The following picture shows the control with the `RenderType` property on `0x4000FFFF` (25% Yellow, `0x40` or 64 in decimal is 25% from 256):



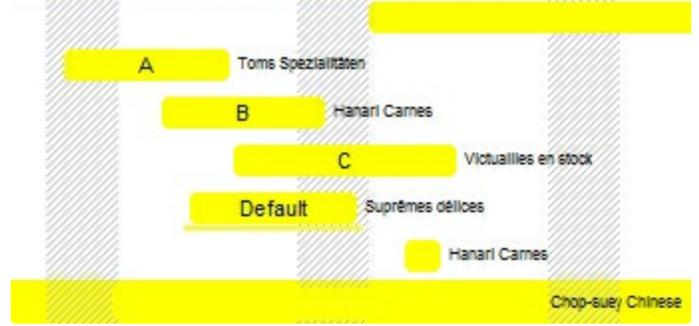
The following picture shows the control with the *RenderType* property on `0x800FFFF` (50% Yellow, `0x80` or 128 in decimal is 50% from 256):



The following picture shows the control with the *RenderType* property on `0xC00FFFF` (75% Yellow, `0xC0` or 192 in decimal is 75% from 256):



The following picture shows the control with the *RenderType* property on `0xFF0FFFF` (100% Yellow, `0xFF` or 255 in decimal is 100% from 255):



Column object

The ExComboBox control supports multiple columns. The Columns object contains a collection of Column objects. By default, the control has no columns, so the user needs to add at least one column, before inserting new items. The Column object holds information about a control's column like: Alignment, Caption, Position and so on. The Column object supports the following properties and methods:

Name	Description
Alignment	Retrieves or sets the alignment of the cells in the column.
AllowDragging	Retrieves or sets a value indicating whether the column can be dragged to a new position using the mouse.
AllowEditContextMenu	Specifies whether the default popup menu of associated edit control is shown when the user does a right click in the label area.
AllowSizing	Retrieves or sets a value indicating whether the user can change the column's width by dragging the column's resize bar.
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	Shows or hides the column's filterbar 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 filterbar 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.

[EditAlignment](#)

Retrieves or sets the alignment of column's edit control.

[EditMaxLength](#)

Gets or sets the number of characters a user can type into the control's edit.

[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](#)

Specifies whether the control fires FormatColumn to format the captions 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 should appear in bold.

[HeaderImage](#)

Specifies the index of image in the Images collection being displayed on 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.

[HTMLCaption](#)

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

Index	Returns a value that indicates the index of the column in the Columns 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.
PartialCheck	Specifies whether the column supports partial check feature.
Position	Retrieves or sets a value that indicates the position of the column in the header bar area.
Prompt	Indicates the developer-specified prompt when the associated edit control contains no text.
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 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 cells in the column.

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

Use the Alignment property to change the alignment of the cells in the column. Use the [HeaderAlignment](#) property to align the column's caption in the control's header. Use the [CellHAlignment](#) property to align a particular cell. Please make sure that If the cell belongs to the column that displays the hierarchy in the control ([TreeColumnIndex](#) property indicates the index of the column that displays the hierarchy lines), the caption, icon or its picture can't be centered. In this case the caption can be aligned to the left or to the right. Use the [HasLines](#) property to display the control's hierarchy lines. Use the [CellHAlignment](#) property to align a particular cell. Use the [RightToLeft](#) property to align the drop down button to the left side of the control. The [RightToLeft](#) property specifies the drawing order of the control's elements.

property Column.AllowDragging as Boolean

Retrieves or sets a value indicating whether the column can be dragged to a new position using the mouse.

Type	Description
Boolean	A boolean expression indicating whether the column can be dragged to a new position using the mouse.

Use the AllowDragging property to forbid user to change the column's position by dragging. Use the [Position](#) property of the Column object to change programmatically the column's position. Use the [AllowSizing](#) property to allow user resizes a column at runtime. Use the [AllowSort](#) property to avoid sorting a column when the user clicks the column's header

property Column.AllowEditContextMenu as Boolean

Specifies whether the default popup menu of associated edit control is shown when the user does a right click in the label area.

Type	Description
Boolean	A boolean expression that specifies whether the associated edit control displays the default popup/context menu when the user does a right click in the label area.

By default, the AllowEditContextMenu property is True. Use the AllowEditContextMenu property to disable the default context menu being displayed when the user right clicks the label area, and provides your own context menu. This property has effect only if the [Style](#) property is DropDown or Simple, as the DropDownList style does not provide an edit control associated to each column. The [RClick](#) event is fired when the AllowEditContextMenu property is False, and the user does a right click inside the edit control displayed on the control's label. Use the [DropDown](#) property to determine whether the drop down portion of the control is shown or hidden.

property Column.AllowSizing as Boolean

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

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

Use the AllowSizing property to fix the column's width. Use the [Width](#) property of the column to change programmatically the column's width. Use the [ColumnAutoResize](#) property to fit all visible columns to the control's client area. Use the [AllowDragging](#) property to forbid user to change the column's position by dragging. Use the [Width](#) property to specify the column's width. Use the [AllowSort](#) property to avoid sorting a column when the user clicks the column's header.

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. */*not supported in the lite version*/*

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 AutoSearch expression that defines the type of incremental searching.

By default, the AutoSearch property is exStartWith. The AutoSearch property has effect only if the control's [Style](#) property is DropDownList, and [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. The searching column is defined by the [SearchColumnIndex](#) property. Use the [ExpandOnSearch](#) property to expand items while user types characters in the control.

property Column.AutoWidth as Long

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

Type	Description
Long	A long expression that indicates the column's width required 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 doesn't change the column's width. Use [Width](#) property to change the column's width at runtime. 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 EXCOMBOBOXLibCtl.ComboBox)
    t.BeginUpdate
        Dim c As Column
        For Each c In t.Columns
            c.Width = c.AutoWidth
        Next
    t.EndUpdate
End Sub
```

The following C++ sample resizes all visible columns:

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

The following VB.NET sample resizes all visible columns:

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

The following C# sample resizes all visible columns:

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

The following VFP sample resizes all visible columns:

```
with thisform.ComboBox1
    .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 the text displayed to the column's header.

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

Adding two columns with the same caption is supported and they can be differentiated by their indexes or their keys. Use the [Key](#) property to assign a key to a column. Use the [HTMLCaption](#) property to use built-in HTML tags in the column's caption. Use the [Add](#) method to add a new column to the control's Columns collection, and to specify the column's caption at adding time. The column's caption is displayed using the following font attributes: [HeaderBold](#), [HeaderItalic](#), [HeaderUnderline](#), [HeaderStrikeout](#). The [HeaderImage](#) property assign an icon to the column's header.

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 CellCaptionFormat 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 [CellCaptionFormat](#) property on exText, uses the same formula to display their content. For instance, you can use the CellCaptionFormat property on exHTML, for cells in the column, that need to display other things than column's formula, or you can use the CellCaptionFormat property on exComputedField, to change the formula for a particular cell. Use the CellCaptionFormat property to change the type for a particular cell. Use the [CellCaption](#) property to specify the cell's content. For instance, if the CellCaptionFormat property is exComputedField, the Caption property indicates the formula to compute the cell's content. The [Def\(exCellCaptionFormat\)](#) 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\(exCellCaptionFormat\)](#) 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 [CellCaption](#) 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 fourth 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. `"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

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]]].

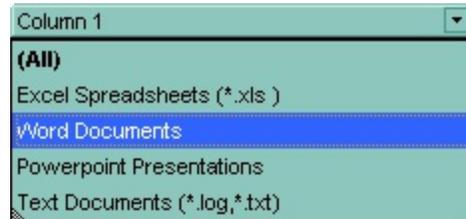
For example, you may have a list of documents and instead of listing the name of each document in the filter drop down list for the names column you may want to list the following:

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 an extra data to a column. The Data property is not used by the control, so you can hold any extra data associated to the column. You can use the [RemoveColumn](#) event to release the extra data that's associated to a column, in case it is required. Use the [ItemData](#) property to associate an extra data to an item. Use the [CellData](#) property to assign an extra data to a cell.

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.

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

```
ComboBox1.Columns(0).Def(exCellHasCheckBox) = True
```

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

```
With ComboBox1.Columns(0)  
    .Def(exCellBackColor) = RGB(240, 240, 240)  
End With
```

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

```
COleVariant vtCheckBox( VARIANT_TRUE );  
m_combobox.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_combobox.GetColumns().GetItem( COleVariant( (long) 0 ) ).SetDef( /*exCellBackColor*/  
4, vtBackColor );
```

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

```
With AxComboBox1.Columns(0)  
    .Def(EXCOMBOBOXLib.DefColumnEnum.exCellHasCheckBox) = True  
End With
```

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

```
With AxComboBox1.Columns(0)
    .Def(EXCOMBOBOXLib.DefColumnEnum.exCellBackColor) =
    ToUInt32(Color.WhiteSmoke)
End With
```

where the ToUInt32 function converts a Color expression to an 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:

```
axComboBox1.Columns[0].set_Def(
EXCOMBOBOXLib.DefColumnEnum.exCellHasCheckBox, true );
```

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

```
axComboBox1.Columns[0].set_Def(EXCOMBOBOXLib.DefColumnEnum.exCellBackColor,
ToUInt32(Color.WhiteSmoke));
```

where the ToUInt32 function converts a Color expression to an 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.ComboBox1.Columns(0)
```

```
  .Def( 0 ) = .t.
```

```
endwith
```

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

```
with thisform.ComboBox1.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 indicates 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 when user can sort the columns by clicking the control's header. The [SortOrder](#) property specifies the column's sort order. Use the [SortChildren](#) method to sort items at runtime.

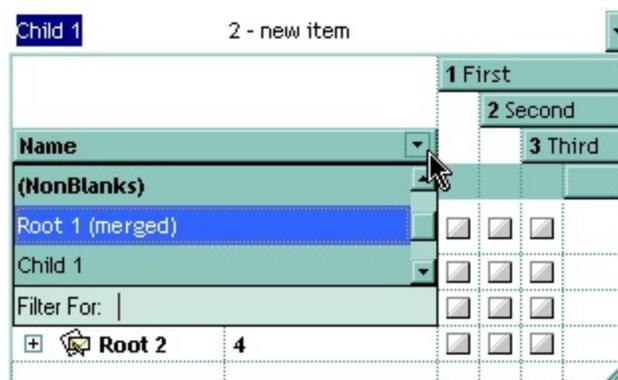
property Column.DisplayFilterButton as Boolean

Shows or hides the column's filter bar button. */*not supported in the lite version*/*

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. 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 the [FilterInclude](#) property to specify whether the child items should be included to the list when the user applies the filter. Use the [Background\(exHeaderFilterBarButton\)](#) property to change the visual appearance for the drop down filter button. Use the [FilterCriteria](#) property to filter items using AND, OR or NOT operators between columns. Use the [CustomFilter](#) property to define you custom filters. Use the [ShowFilter](#) method to show programmatically the column's drop down filter window. The [FilterForVisible](#) property specifies whether the drop down portion of the control displays all the time a text-box editor to allow filtering items of the control.

For instance, if the user clicks the filter bar button, the control displays a drop down window where all values from the column are inserted like in the following screen shot:



property Column.DisplayFilterDate as Boolean

Specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for. */*not supported in the lite version*/*

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.

*/*not supported in the lite version*/*

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 DisplayFilterPattern 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.

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 the column's header, while the column is sorted.

Use the DisplaySortIcon property to hide the sort icon. Use the [SortOnClick](#) property of control to disable sorting columns by clicking in the column's header.

The control automatically sorts a column when the user clicks the column's header, if the SortOnClick property is exDefaultSort. If the SortOnClick property is exNoSort, the control disables sorting the items when the user clicks the column's header. There are two methods to get the items sorted like follows:

- 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:

```
ComboBox1.Columns(ColIndex).SortOrder = SortAscending
```

- Using the [SortChildren](#) method of the [Items](#) object. The SortChildren sorts the items. The SortChildren method sorts the child items of the given parent item in the control. SortChildren will not recourse through the tree, only the immediate children of Item will be sorted. The following sample sort descending the list of root items on the "Column 1"(if your control displays a list, all items are considered being root items).

```
ComboBox1.Items.SortChildren 0, "Column 1", False
```

property Column.EditAlignment as AlignmentEnum

Retrieves or sets the alignment of column's edit control.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the column's edit alignment.

Use the EditAlignment property to align the caption in the edit control of the column. By default, the EditAlignment property is LeftAlignment. Use the [HeaderAlignment](#) property to align the caption of the column in the control's header bar. Use the [Alignment](#) property to align cells inside a column. Use the [EditText](#) property to retrieve the text in the control's label. Use the [Select](#) property to get or set the selected value when control contains multiple columns.

property Column.EditMaxLength as Long

Gets or sets the number of characters a user can type into the control's edit.

Type	Description
Long	A long expression that defines the maximum characters the user can type into the column's editor.

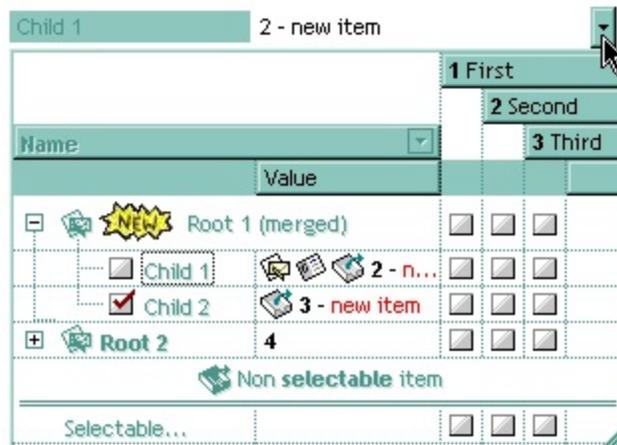
By default, the EditMaxLength property is 0, which indicates no effect. The EditMaxLength property limits the number of characters the user can type into the column's text-box. The control display no edit-controls if the [Style](#) property is DropDownList. The [AutoComplete](#) property disables disable auto complete feature. The [AutoSearch](#) property disables the control's incremental search feature. For instance, EditMaxLength property on 2 limits the user to type up to 2 characters within the giving column.

property Column.Enabled as Boolean

Returns or sets a value that determines whether a column's header can respond to user-generated events.

Type	Description
Boolean	A boolean expression that determines whether a column's header can respond to user-generated events.

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. A disabled column looks grayed. If the column is disabled the associated edit control (if it is exist) in the control's label is disabled too. Use the [Style](#) property to specify the style of the control.



The following VB sample disables the first column:

```
ComboBox1.Columns(0).Enabled = False
```

The following C++ sample disables the first column:

```
m_combobox.GetColumns().GetItem(COLEVariant(long(0))).SetEnabled( FALSE );
```

The following VB.NET sample disables the first column:

```
With AxComboBox1.Columns(0)  
    .Enabled = False  
End With
```

The following C# sample disables the first column:

```
axComboBox1.Columns[0].Enabled = false;
```

The following VFP sample disables the first column:

```
with thisform.ComboBox1.Columns(0)
    .Enabled = .f.
endwith
```

property Column.Filter as String

Specifies the column's filter when the filter type is `exFilter`, `exPattern`, `exDate`, `exNumeric`, `exCheck` or `exImage`. */*not supported in the lite version*/*

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 (globbing). 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. The [CustomFilter](#) property has no effect of the FilterType property is `exNumeric`.
- 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. The [CustomFilter](#) property has no effect of the FilterType property is exCheck.

- 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 [CustomFilter](#) property has no effect of the FilterType property is exImage.

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 [FilterInclude](#) property to specify whether the child items should be included to the list when the user applies the filter. Use the [FilterCriteria](#) property to filter items using AND, OR or NOT operators between columns. The [FilterForVisible](#) property specifies whether the drop down portion of the control displays all the time a text-box editor to allow filtering items of the control.

The following VB sample filters "Child 1" and "Child 2" items:

```
With ComboBox1
  With .Columns(0)
    .Filter = "Child 1|Child 2"
    .FilterType = exPattern
  End With
  .ApplyFilter
End With
```

The following C++ sample filters "Child 1" and "Child 2" items:

```
CColumn column = m_combobox.GetColumns().GetItem(COleVariant(long(0)));
column.SetFilter( "Child 1|Child 2" );
column.SetFilterType( 3 /*exPattern*/ );
m_combobox.ApplyFilter();
```

The following VB.NET sample filters "Child 1" and "Child 2" items:

```
With AxComboBox1
```

```
  With .Columns(0)
```

```
    .Filter = "Child 1|Child 2"
```

```
    .FilterType = EXCOMBOBOXLib.FilterTypeEnum.exPattern
```

```
  End With
```

```
  .ApplyFilter()
```

```
End With
```

The following C# sample filters "Child 1" and "Child 2" items:

```
axComboBox1.Columns[0].Filter = "Child 1|Child 2";
```

```
axComboBox1.Columns[0].FilterType = EXCOMBOBOXLib.FilterTypeEnum.exPattern;
```

```
axComboBox1.ApplyFilter();
```

The following VFP sample filters "Child 1" and "Child 2" items:

```
With thisform.ComboBox1
```

```
  With .Columns.Item(0)
```

```
    .Filter = "Child 1|Child 2"
```

```
    .FilterType = 3 && exPattern
```

```
  EndWith
```

```
  .ApplyFilter
```

```
EndWith
```

property Column.FilterBarDropDownWidth as Double

Specifies the width of the drop down filter window proportionally with the width of the column. */*not supported in the lite version*/*

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. The FilterBarDropDownWidth property is changed, if the user resizes at run-time the drop down filter window. 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 ComboBox1.Columns(0)
    .FilterBarDropDownWidth = 2
End With
```

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

```
With ComboBox1.Columns(0)
    .FilterBarDropDownWidth = -150
End With
```

property Column.FilterList as FilterListEnum

Specifies whether the drop down filter list includes visible or all items. */*not supported in the lite version*/*

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.

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. User 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:

A	B	A+B
Group 1		
16	17	33
2	11	13
2	9	11
Group 2		
16	9	25
12	11	23
2	2	4
Group 1		
16	17	33

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. */*not supported in the lite version*/*

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 [FilterInclude](#) property to specify whether the child items should be included to the list when the user applies the filter. Use the [FilterCriteria](#) property to filter items using AND, OR or NOT operators between columns. The [FilterForVisible](#) property specifies whether the drop down portion of the control displays all the time a text-box editor to allow filtering items of the control.

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.

The following VB sample filters "Child 1" and "Child 2" items:

```
With ComboBox1
  With .Columns(0)
    .Filter = "Child 1|Child 2"
    .FilterType = exPattern
  End With
  .ApplyFilter
End With
```

The following C++ sample filters "Child 1" and "Child 2" items:

```
CColumn column = m_combobox.GetColumns().GetItem(COleVariant(long(0)));
column.SetFilter( "Child 1|Child 2" );
column.SetFilterType( 3 /*exPattern*/ );
m_combobox.ApplyFilter();
```

The following VB.NET sample filters "Child 1" and "Child 2" items:

```
With AxComboBox1
```

```
With .Columns(0)
    .Filter = "Child 1|Child 2"
    .FilterType = EXCOMBOBOXLib.FilterTypeEnum.exPattern
End With
.ApplyFilter()
End With
```

The following C# sample filters "Child 1" and "Child 2" items:

```
axComboBox1.Columns[0].Filter = "Child 1|Child 2";
axComboBox1.Columns[0].FilterType = EXCOMBOBOXLib.FilterTypeEnum.exPattern;
axComboBox1.ApplyFilter();
```

The following VFP sample filters "Child 1" and "Child 2" items:

```
With thisform.ComboBox1
    With .Columns.Item(0)
        .Filter = "Child 1|Child 2"
        .FilterType = 3 && exPattern
    EndWith
    .ApplyFilter
EndWith
```

property Column.FireFormatColumn as Boolean

Specifies whether the control fires FormatColumn to format the captions in the column.

Type	Description
Boolean	A boolean expression that indicates whether the control fires the FireFormatColumn event for the cells in the column.

By default, the FireFormatColumn property is False. The [FormatColumn](#) event is fired only if the FireFormatColumn property of the Column object 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 \$50 instead 50. Also, it is useful to use the FormatColumn event when displaying computed cells.

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 [CellCaptionFormat](#) or [Def\(exCellCaptionFormat\)](#) 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>' : '') + 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
Root				
Child 1	7+	3+	1=	\$11.00
Child 2	2+	6+	12=	\$19.00
Child 3	2+	2+	4=	\$8.00
Child 4	2+	9+	4=	\$15.00

The **value** keyword in the FormatColumn 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 [CellCaption](#) 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 ""

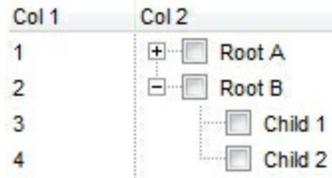
Col 1	Col 2
1	+ Root A
4	- Root B
5	Child 1
6	Child 2

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

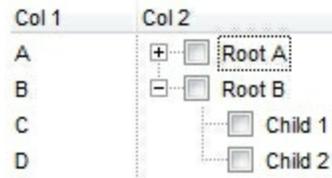
Col 1	Col 2
A	+ Root A
D	- Root B
E	Child 1
F	Child 2

- 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 ""`

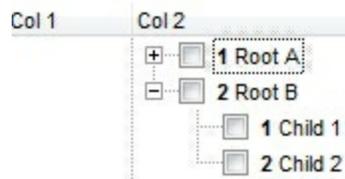


In the following screen shot the `FormatColumn("Col 1") = "1 apos 'A-Z'"`

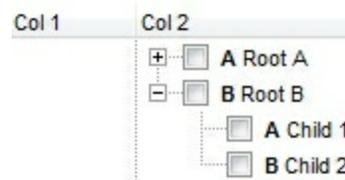


- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance, `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 ""

Col 1	Col 2
1	+ Root A
2	- Root B
2.1	Child 1
2.2	Child 2

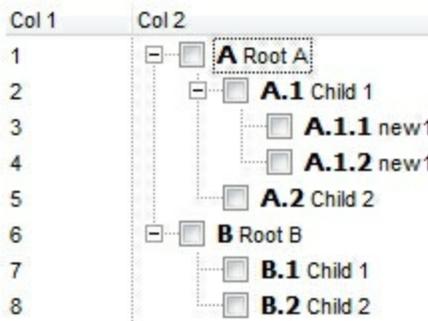
In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"

Col 1	Col 2
A	+ Root A
B	- Root B
B:A	Child 1
B:B	Child 2

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

Col 1	Col 2
A	- Root A
A.1	Child 1
A.2	Child 2
B	- Root B
B.1	Child 1
B.2	Child 2

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 ComboBox1

```
.Columns.Add("Currency").FormatColumn = "currency(dbl(value))"
```

With .Items

```
.AddItem "1.23"
```

```
.AddItem "2.34"
```

```
.AddItem "0"
```

```
.AddItem 5
```

```
.AddItem "10000.99"
```

End With

End With

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

With AxComboBox1

```
.Columns.Add("Currency").FormatColumn = "currency(dbl(value))"
```

With .Items

```
.AddItem "1.23"
```

```
.AddItem "2.34"
```

```
.AddItem "0"
```

```
.AddItem 5
```

```
.AddItem "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 'EXCOMBOBOXLib' for the library: 'ExComboBox 1.0 Control  
    Library'  
  
    #import "C:\\Windows\\System32\\ExComboBox.dll"  
    using namespace EXCOMBOBOXLib;  
*/  
EXCOMBOBOXLib::IComboBoxPtr spComboBox1 = GetDlgItem(IDC_COMBOBOX1)-  
>GetControlUnknown();  
((EXCOMBOBOXLib::IColumnPtr)(spComboBox1->GetColumns()->Add(L"Currency")))-  
>PutFormatColumn(L"currency(dbl(value))");  
EXCOMBOBOXLib::IItemsPtr var_Items = spComboBox1->GetItems();  
var_Items->AddItem("1.23");  
var_Items->AddItem("2.34");  
var_Items->AddItem("0");  
var_Items->AddItem(long(5));  
var_Items->AddItem("10000.99");
```

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

```
(axComboBox1.Columns.Add("Currency") as EXCOMBOBOXLib.Column).FormatColumn  
= "currency(dbl(value))";  
EXCOMBOBOXLib.Items var_Items = axComboBox1.Items;  
var_Items.AddItem("1.23");  
var_Items.AddItem("2.34");  
var_Items.AddItem("0");  
var_Items.AddItem(5);  
var_Items.AddItem("10000.99");
```

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

```
with thisform.ComboBox1  
.Columns.Add("Currency").FormatColumn = "currency(dbl(value))"  
with .Items  
.AddItem("1.23")
```

```
.AddItem("2.34")
```

```
.AddItem("0")
```

```
.AddItem(5)
```

```
.AddItem("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 for the column's caption.

The HeaderAlignment aligns the column's caption. The HeaderAlignment property doesn't align the cells in the column. Use the [Alignment](#) property to align the cells in the column. Use the [CellHAlignment](#) property to align only a cell. Use the [HeaderImageAlignment](#) property to align the column's icon inside the column's header. The [RightToLeft](#) property specifies the drawing order of the control's elements.

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 bolds the column's caption. Use the tag in the [HTMLCaption](#) property to display multiple parts using bold font attribute. The [CellBold](#) property bolds a cell. The [ItemBold](#) property bolds an item. The column's caption is displayed using the following font attributes: HeaderBold, [HeaderItalic](#), [HeaderUnderline](#), [HeaderStrikeOut](#).

property Column.HeaderImage as Long

Specifies the index of image in the Images collection being displayed on the column's header.

Type	Description
Long	A long expression that indicates the index of the image 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.

The HeaderImage property assign an icon to the column's header. Use the [Images](#) method to assign a list of icons to the control. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. Use the [HeaderImageAlignment](#) property to align the icon in the column's header. Use the [CellImage](#) property to assign an icon to a cell. Use the [CellImages](#) property to assign multiple icons to a cell. Use the [Caption](#) property to assign a new caption to a column.

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. The [RightToLeft](#) property specifies the drawing order of the control's elements.

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.

The HeaderItalic property specifies whether the column's caption should appear in italic. Use the <i> tag in the [HTMLCaption](#) property to display multiple parts using italic font attribute. The [CellItalic](#) property specifies whether the cell should appear in italic. The [ItemItalic](#) property specifies whether the item should appear in italic. The column's caption is displayed using the following font attributes: [HeaderBold](#), HeaderItalic, [HeaderUnderline](#), [HeaderStrikeOut](#).

property Column.HeaderStrikeOut as Boolean

Retrieves or sets the StrikeOut property of the Font object that it is used to paint the column's caption.

Type	Description
Boolean	A boolean expression that indicates whether the column's header is strikouted.

The HeaderStrikeOut property specifies whether the column's caption should appear in strikeout. Use the <s> tag in the [HTMLCaption](#) property to display multiple parts using strikeout font attribute. The [CellStrikeOut](#) property specifies whether the cell should appear in strieout. The [ItemStrikeOut](#) property specifies whether the item should appear in strikeout. The column's caption is displayed using the following font attributes: [HeaderBold](#), [HeaderItalic](#), [HeaderUnderline](#), HeaderStrikeOut.

property Column.HeaderUnderline as Boolean

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

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

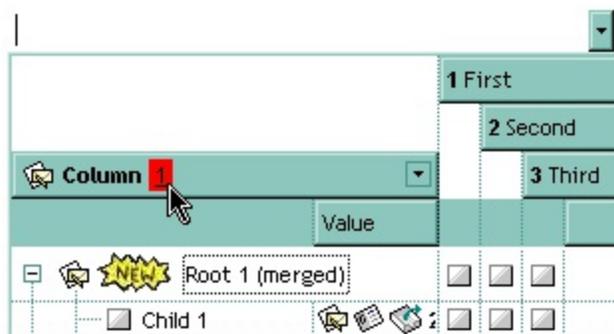
The HeaderUnderline property specifies whether the column's caption is underlined. Use the <u> tag in the [HTMLCaption](#) property to display multiple parts using underline font attribute. The [CellUnderline](#) property specifies whether the cell is underlined. The [ItemUnderline](#) property specifies whether the item is underlined. The column's caption is displayed using the following font attributes: [HeaderBold](#), [HeaderItalic](#), HeaderUnderline, [HeaderStrikeOut](#).

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. The list of built-in HTML tags supported are [here](#). Use the [CellCaptionFormat](#) property to allow built-in HTML format in cells. Use the `` HTML tag to insert icons inside the column's caption.



The following VB sample assign a HTML caption to the first column:

```
With ComboBox1.Columns(0)
    .HTMLCaption = "<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor>"
End With
```

The following C++ sample assign a HTML caption to the first column:

```
m_combobox.GetColumns().GetItem(COLEVariant(long(0))).SetHTMLCaption( "  
<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor> " );
```

The following VB.NET sample assign a HTML caption to the first column:

```
With AxComboBox1.Columns(0)
    .HTMLCaption = "<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor>"
End With
```

The following C# sample assign a HTML caption to the first column:

```
axComboBox1.Columns[0].HTMLCaption = "<b>Column</b> <bgcolor=FF0000>
```

```
<u>1</u> </bgcolor>";
```

The following VFP sample assign a HTML caption to the first column:

```
with thisform.ComboBox1.Columns(0)
```

```
  .HTMLCaption = "<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor> "  
endwith
```

property Column.Index as Long

Returns a value that indicates the index of the column in the Columns collection.

Type	Description
Long	A long expression that indicates the index of the column in the Columns collection.

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 indicates the column's key.

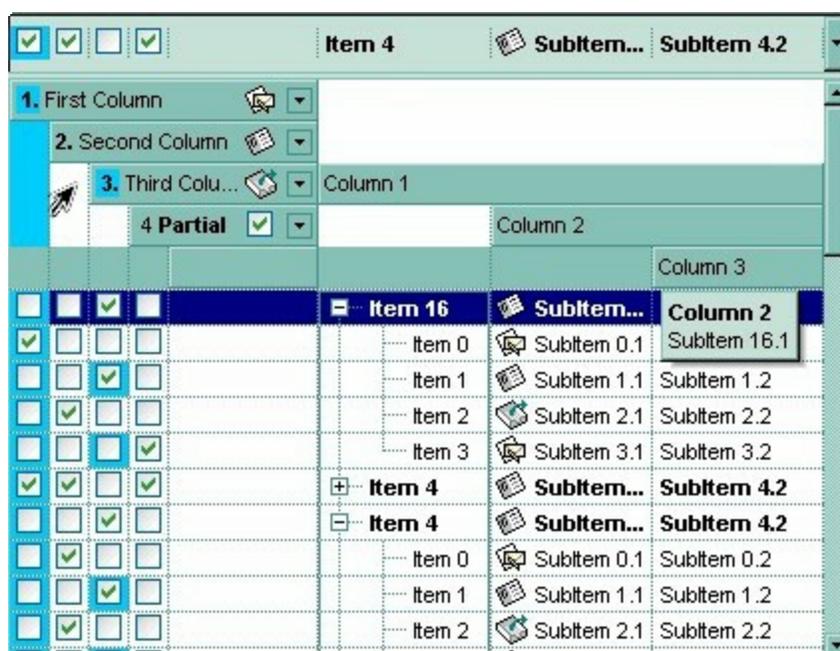
A column is identifies by its caption, key or index. The column's key defines a column when using the [Item](#) property. Use the [Index](#) or the Key property to identify a column, when using the [Columns](#) property.

property Column.LevelKey as Variant

Retrieves or sets a value that indicates the key of the column's level. */*not supported in the lite version*/*

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

By default, the LevelKey property 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.



property Column.MaxWidthAutoSize as Long

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

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

Use the MaxWidthAutoSize property to set the maximum column's width while the [WidthAutoSize](#) property is True. If the MaxWidthAutoSize property is less than zero, there is no maximum value for the column's width. By default, the MaxWidthAutoSize property is -1. Use the [ColumnAutoSize](#) property to specify whether the control resizes the visible columns so they fit the control's client area.

property Column.MinWidthAutoSize as Long

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

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

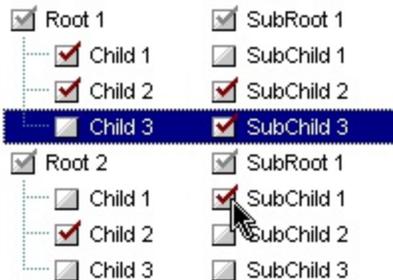
Use the MinWidthAutoSize property to set the minimum column's width while the [WidthAutoSize](#) property is True. Use the [Width](#) property to specify the column's width. Use the [ColumnAutoSize](#) property to specify whether the control resizes the visible columns so they fit the control's client area.

property Column.PartialCheck as Boolean

Specifies whether the column supports partial check feature.

Type	Description
Boolean	A boolean expression that indicates whether the control supports the partial check feature.

The PartialCheck property specifies that the column supports partial check feature. By default, the PartialCheck property is False. Use the [CellHasCheckBox](#) property to associate a check box to a cell. Use the [Def](#) property to assign a cell box for the entire column. Use the [CellState](#) property to determine the cell's state. If the PartialCheck property is True, the CellState property has three states: 0 - Unchecked, 1 - Checked and 2 - Partial Checked. Use the [CheckImage](#) property to define the icons for each state. The control supports partial check feature for any column that your control contains. Use the [Add](#) method to add new columns to the control.



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 programmatically the column's position. The user is able to change the column's position by dragging its caption to another location. If the [AllowDragging](#) property is False, the column's position cannot be changed by dragging. Use the [Visible](#) property to hide a column. Use the [Width](#) property to specify the column's width. The [RightToLeft](#) property specifies the drawing order of the control's elements.

property Column.Prompt as String

Indicates the developer-specified prompt when the associated edit control contains no text.

Type	Description
String	A String expression that specifies the prompt is a label or short instruction placed inside an editable drop-down list as its default value.

By default, the Prompt property is empty. The Prompt gets displayed only if the control's [Style](#) property is DropDown or Simple (as an editable area is provided in the control's label). A prompt is a label or short instruction placed inside an editable drop-down list as its default value. Unlike static text, prompts disappear from the screen once users type something into the combo box or it gets input focus. Use the [SingleEdit](#) property to specify whether the control's label displays a single or multiple columns. The [EditText](#) property to change the text being displayed in the control's label, while the Style property is DropDown or Simple.

Use a prompt when:

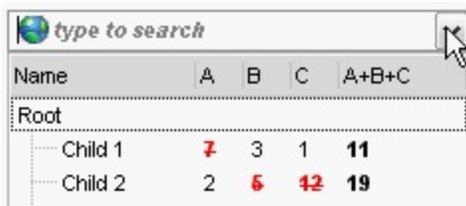
- The prompt is primarily for identifying the purpose of the list in a compact way. It must not be crucial information that users need to see while using the combo box.
- For example, prompts like *Select an option* or *Enter a filename and then click Send* are unnecessary.
- The prompt text must not be confused with real text, so you can use the `<fgcolor>` HTML tag to specify a different foreground color.

For instance, the following Prompt "`<fgcolor=808080><i>type to search<r>1`" is showing as follows , and so the image gets aligned to the right due `<r>` field, and the text gets aligned to the left.

If you are changing the column's prompt at runtime, you need to call the `EditText` property as follows, else the prompt will not be updated until a change occurs:

```
With ComboBox1
    .Columns(0).Prompt = "<img>1</img> <i> <fgcolor=808080>type to
search</fgcolor> </i>"
    .EditText(0) = ""
End With
```

The following screen shot shows how the prompt is disappearing as soon as the user starts typing characters, and a new item gets selected:



Name	A	B	C	A+B+C
Root				
Child 1	7	3	1	11
Child 2	2	6	12	19

The Prompt property supports the built-in HTML tags listed [here](#).

method Column.ShowFilter ([Options as Variant])

Shows the column's filter window.

Type

Description

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:

Options as Variant

- 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 ComboBox1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If (Button = 2) Then
        With ComboBox1.Columns
            With .Item(ComboBox1.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 AxComboBox1_MouseUpEvent(ByVal sender As Object, ByVal e As AxEXCOMBOBOXLib._IComboBoxEvents_MouseUpEvent) Handles AxComboBox1.MouseUpEvent
    If (e.button = 2) Then
        With AxComboBox1.Columns
            With .Item(AxComboBox1.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 axComboBox1_MouseUpEvent(object sender, AxEXCOMBOBOXLib._IComboBoxEvents_MouseUpEvent e)
{
    if (e.button == 2)
    {
        EXCOMBOBOXLib.Column c =
axComboBox1.Columns[axComboBox1.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 OnMouseUpComboBox1(short Button, short Shift, long X, long Y)  
{  
    m_combobox.GetColumns().GetItem( COleVariant( m_combobox.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.ComboBox1.Columns  
        With .Item(thisform.ComboBox1.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.

By default, the control automatically sort a column when the user clicks the column's header. Use the [DefaultSortOrder](#) property to specify the column's default sort order, when the user first clicks the column's header. Use the [SortType](#) property to determine the way how the column is sorted. Use the [DisplaySortIcon](#) property to hide the sort icon if the column is sorted.

The control automatically sorts a column when the user clicks the column's header, if the [SortOnClick](#) property is exDefaultSort. If the SortOnClick property is exNoSort, the control disables sorting the items when the user clicks the column's header. There are two methods to get the items sorted like follows:

- 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:

```
ComboBox1.Columns(ColIndex).SortOrder = SortAscending
```

- Using the [SortChildren](#) method of the [Items](#) object. The SortChildren sorts the items. The SortChildren method sorts the child items of the given parent item in the control. SortChildren will not recourse through the tree, only the immediate children of Item will be sorted. The following sample sort descending the list of root items on the "Column 1"(if your control displays a list, all items are considered being root items).

```
ComboBox1.Items.SortChildren 0, "Column 1", False
```

property Column.SortPosition as Long

Returns or sets a value that indicates the position of the column in the sorting columns collection. */*not supported in the lite version*/*

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 a control sorts the values for a column.

Type	Description
SortTypeEnum	A SortTypeEnum expression that indicates the way a control sorts the values for a column.

By default, the column's SortType is String. Use the SortType property to specifies how the control will sort the column. Use the [SortChildren](#) property of Items to do a sort based on a column. The [SortOrder](#) property determines the column's sort order.

property Column.ToolTip as String

Specifies the column's tooltip description.

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

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. The control fires the [ToolTip](#) event when the column's tooltip is about to be displayed. 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 period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [CellToolTip](#) property to assign a tooltip to a cell.

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 change the column's width. Use the [AdjustSearchColumn](#) property to allow displaying a hidden column in the control's label area. The [SearchColumnIndex](#) property indicates the index of the column being displayed in the control's label area if the [SingleEdit](#) property is `True`. Use the [CellOwnerDraw](#) property to draw a cell. 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 width of the column in pixels. Use the [Visible](#) property to hide a column. Use the [ColumnAutoResize](#) property to specify whether the visible columns should fit the control's client area. Use the [Position](#) property to specify the column's position. Use the [SingleEdit](#) property to specify whether the control's label displays one or multiple columns. 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.

The following VB sample sets the width for all columns:

```
Private Sub ComboBox1_AddColumn(ByVal Column As EXCOMBOBOXLibCtl.IColumn)
    Column.Width = 128
End Sub
```

The following C++ sample sets the width for all columns:

```
#include "Column.h"
void OnAddColumnCombobox1(LPDISPATCH Column)
{
    CColumn column( Column );column.m_bAutoRelease = FALSE;
    column.SetWidth( 128 );
}
```

The following VB.NET sample changes the column's width:

```
Private Sub AxComboBox1_AddColumn(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_AddColumnEvent) Handles
AxComboBox1.AddColumn
    e.column.Width = 128
End Sub
```

The following C# sample changes the column's width:

```
private void axComboBox1_AddColumn(object sender,
```

```
AxEXCOMBOBOXLib._IComboBoxEvents_AddColumnEvent e)
```

```
{  
    e.column.Width = 128;  
}
```

The following VFP sample changes the column's width:

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

```
LPARAMETERS column
```

```
with column
```

```
    .Width = 128
```

```
endwith
```

property Column.WidthAutoSize 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 `WidthAutoSize` property is `True`, the column's width is resized after user expands, or collapse the items. Also, the column's width is refreshed if the user adds new items to the control. If the `WidthAutoSize` property is `True`, the column's width is not larger than [MaxWidthAutoSize](#) value, and it is not less than [MinWidthAutoSize](#) 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 [ColumnAutoSize](#) property to specify whether the control resizes the visible columns so they fit the control's client area.

Columns object

The ExComboBox control supports more columns. The Columns object contains a collection of Column objects. Use the Columns property of the control to access the control columns. By default, the control doesn't add any new column to the collection, so before inserting any new items you have to be sure that you have already added at least one column.

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 indicates the column's caption.
Return	Description
Variant	A Column object that represents the newly added column.

Use the [Caption](#) property to change the column's caption. Use the [HTMLCaption](#) property to assign a HTML caption to a column. The [AddColumn](#) event is fired when the user adds a new column to the Columns collection. Use the [Remove](#) method to remove a column. Use the [Data](#) property to assign an extra data to a column. Use the [AddItem](#), [InsertItem](#), [PutItems](#), [DataSource](#) properties to add new items to the control. Use the [BeginUpdate](#) and [EndUpdate](#) methods to maintain performance while adding new columns and items. Use the [ColumnAutoResize](#) property to specify whether the visible columns fit the control's client area.

The following VB sample adds columns to your control based on 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
ComboBox1.BeginUpdate
For Each f In rs.Fields
    ComboBox1.Columns.Add f.Name
Next
ComboBox1.EndUpdate
```

The following VB sample changes the column's caption to display built-in HTML tags:

```
With ComboBox1
    With .Columns
        With .Add("Column 1")
            .HTMLCaption = "Column <b>1 </b>"
        End With
    End With
End With
```

The following C++ sample adds a column that displays built-in HTML format in its header:

```
CColumn column( V_DISPATCH( &m_combobox.GetColumns().Add( "Column 1" ) ) );  
column.SetHTMLCaption( "Column <b>1</b>" );
```

The following VB.NET sample adds a column that displays built-in HTML format in its header:

```
With AxComboBox1.Columns
```

```
    With .Add("1")
```

```
        .HTMLCaption = "<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor>"
```

```
    End With
```

```
End With
```

The following C# sample adds a column that displays built-in HTML format in its header:

```
EXCOMBOBOXLib.Column column = axComboBox1.Columns.Add("1") as
```

```
EXCOMBOBOXLib.Column;
```

```
column.HTMLCaption = "<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor>";
```

The following VFP sample adds a column that displays built-in HTML format in its header:

```
with thisform.ComboBox1.Columns
```

```
    with .Add("1")
```

```
        .HTMLCaption = "<b>Column</b> <bgcolor=FF0000> <u>1</u> </bgcolor>"
```

```
    endwith
```

```
endwith
```

method Columns.Clear ()

Removes all objects in a collection.

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

Use the [Remove](#) method when you need to remove only a column. Use the Clear method when you need to clear the entire columns collection. When the Clear method is called, the control removes automatically all items too. To remove all items call [RemoveAllItems](#) property of the Items object. The [RemoveColumn](#) event is fired when the user removes a column.

property Columns.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	A long expression that counts the Column objects into 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 ComboBox1.Columns
    Debug.Print c.Caption
Next
```

The following VB sample enumerates the columns in the control:

```
For i = 0 To ComboBox1.Columns.Count - 1
    Debug.Print ComboBox1.Columns(i).Caption
Next
```

The following VC sample enumerates the columns in the control:

```
#include "Columns.h"
#include "Column.h"
CColumns columns = m_combobox.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 AxComboBox1.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:

```
EXCOMBOBOXLib.Columns columns = axComboBox1.Columns;
for ( int i = 0; i < columns.Count; i++ )
{
    EXCOMBOBOXLib.Column column = columns[i];
    System.Diagnostics.Debug.WriteLine( column.Caption );
}
```

The following VFP sample enumerates the columns in the control:

```
with thisform.ComboBox1.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, a string expression that indicates the column's caption or the columns' 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 Item property is the default property of the Columns object so the following statements are equivalent:

```
ComboBox1.Columns.Item ("Name")  
ComboBox1.Columns ("Name")
```

The following VB sample enumerates the columns in the control:

```
For Each c In ComboBox1.Columns  
    Debug.Print c.Caption  
Next
```

The following VB sample enumerates the columns in the control:

```
For i = 0 To ComboBox1.Columns.Count - 1  
    Debug.Print ComboBox1.Columns(i).Caption  
Next
```

The following VC sample enumerates the columns in the control:

```
#include "Columns.h"  
#include "Column.h"  
CColumns columns = m_combobox.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 AxComboBox1.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:

```
EXCOMBOBOXLib.Columns columns = axComboBox1.Columns;  
for ( int i = 0; i < columns.Count; i++ )  
{  
  EXCOMBOBOXLib.Column column = columns[i];  
  System.Diagnostics.Debug.WriteLine( column.Caption );  
}
```

The following VFP sample enumerates the columns in the control:

```
with thisform.ComboBox1.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. */*not supported in the lite version*/*

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 ComboBox1.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_combobox.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 ) );
}
OutputDebugString( strOutput );

```

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

```

With AxComboBox1
    Dim s As String, i As Integer, c As EXCOMBOBOXLib.Column
    i = 0
    With AxComboBox1.Columns
        c = .ItemBySortPosition(i)
        While (Not c Is Nothing)
            s = s + """" & c.Caption & """" " & If(c.SortOrder =
EXCOMBOBOXLib.SortOrderEnum.SortAscending, "A", "D") & " "
            i = i + 1
            c = .ItemBySortPosition(i)
        End While
    End With
    s = "Sort: " & s
    Debug.WriteLine(s)
End With

```

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

```

string strOutput = "";
int i = 0;
EXCOMBOBOXLib.Column column = axComboBox1.Columns.get_ItemBySortPosition( i );
while ( column != null )
{
    strOutput += column.Caption + " " + ( column.SortOrder ==
EXCOMBOBOXLib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
    column = axComboBox1.Columns.get_ItemBySortPosition( ++i );
}
Debug.WriteLine( strOutput );

```

The following VFP sample displays the list of columns being sorted (the code is listed in the Sort event) :

```
local s, i, c
```

```
i = 0
```

```
s = ""
```

```
With thisform.ComboBox1.Columns
```

```
  c = .ItemBySortPosition(i)
```

```
  do While (!isnull(c))
```

```
    with c
```

```
      s = s + "" + .Caption
```

```
      s = s + " " + If(.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, a string expression that indicates the column's caption or the columns' key.

The Remove method removes a column from the Columns collection. The [RemoveColumn](#) event is called when user removes a column. Use [Clear](#) method when you need to remove all column objects. Use the [Visible](#) property to hide a column. Use the [RemoveItem](#) to remove an item.

ComboBox 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: {CF170E7A-4391-44BD-8D93-29F8D2801EF7}. The object's program identifier is: "Exontrol.ComboBox". The /COM object module is: "ExComboBox.dll"

Built from the ground up using 100% ATL-based code, the ExComboBox represents some of the most advanced combobox technology available in the Component marketplace. The ExComboBox has more things that you can imagine: ADO, DAO support, multiple columns, incremental search feature, mouse wheel support, locked and unlocked columns, multiple edit controls and more. It acts like a combobox and has three styles: Simple, DropDown and DropDownList. The dropped list control can acts like a simple list box, like a list with multiple columns, like a standard tree or like a multicolumn tree control. Every cell or item can apply different font attributes, background or foreground color, can display a caption using more rows, can display an icon, and more. Cut down your development time and add value to your application with this intelligent, easy to use search control. The ComboBox object supports the following properties and methods:

Name	Description
AdjustSearchColumn	Returns or sets a property that indicates whether the SearchColumnIndex could point to a hidden column.
Alignment	Retrieves or sets a value that specifies the alignment of the drop down list.
AllowHResize	Specifies whether the user can resize the width of the drop down portion when AllowSizeGrip is True.
AllowSizeGrip	Retrieves or sets a value indicating whether the drop-down window is resizable at runtime.
AllowVResize	Specifies whether the user can resize the height of the drop down portion when AllowSizeGrip is True.
AnchorFromPoint	Retrieves the identifier of the anchor from point.
Appearance	Retrieves or sets the control's appearance.
ApplyFilter	Applies the filter.
ASCIILower	Specifies the set of lower characters.
ASCIIUpper	Specifies the set of upper characters.
AssignEditImageOnSelect	Assigns the edit's icon when the selection is changed.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoComplete	Retrieves or sets a value that indicates whether auto-complete feature is enabled or disabled.

AutoDrag	Gets or sets a value that indicates the way the component supports the AutoDrag feature.
AutoDropDown	Retrieves or sets a value that indicates whether the control's list automatically drops down once the user starts type into it.
AutoSearch	Retrieves or sets a value that indicates whether auto-search feature is enabled or disabled.
AutoSelect	Retrieves or sets a value indicating whether the control selects the portion of text that doesn't match with the selected item.
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.
BackColorEdit	Specifies the control's edit 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.
Background	Returns or sets a value that indicates the 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 image used by cells of checkbox type.
ClearFilter	Clears the filter.
CloseOnClick	Specifies whether the user closes the drop down portion of the control, by single click.
CloseOnDbClick	Specifies whether the user closes the drop down window by dbl click.
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.
Copy	Copies the control's content to the clipboard, in the EMF format.
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.
DrawGridLines	Retrieves or sets a value that indicates whether the grid lines are visible or hidden.
DropDown	Specifies whether the drop down is visible or hidden.
DropDownBorder	Specifies type of the border for the drop down portion.
DropDownButtonWidth	Retrieves or sets the width, in pixels, to display the drop down button of the control.
EditImage	Specifies a value that indicates the index of icon being displayed on the column's edit box.
EditText	Retrieves or sets a value that indicates the edit's caption for a given column.
Enabled	Enables or disables the control.
EndUpdate	Resumes painting the control after painting is suspended by the BeginUpdate method.
EnsureOnSort	Specifies whether the control ensures that the focused item fits the control's client area, when the user sorts the items.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
ExpandOnSearch	Expands items automatically while user types characters to search for a specific item.

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 filterbar 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.
FilterForBackColor	Retrieves or sets a value that indicates the FilterFor's background color.
FilterForForeColor	Retrieves or sets a value that indicates the FilterFor's foreground color.
FilterForVisible	Specifies whether the control's FilterFor field is shown or hidden.
FilterInclude	Specifies the items being included after the user applies the filter.
FireClickOnSelect	Fires Click event when user changes the selection.
FireNotInList	Specifies whether the control fires the NotInList event.
Font	Retrieves or sets the control's font.
ForeColor	Retrieves or sets a value that indicates the control's foreground color.
ForeColorEdit	Specifies the control's edit 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,

[GridLineStyle](#)

Specifies the style for gridlines in the list part of the control.

[HasButtons](#)

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

[HasButtonsCustom](#)

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

[HasLines](#)

Enhances the graphic representation of a tree control's hierarchy by drawing lines that link child items to their corresponding parent item.

[HeaderAppearance](#)

Retrieves or sets a value that indicates the header's appearance.

[HeaderBackColor](#)

Specifies the header's background color.

[HeaderForeColor](#)

Specifies the header's foreground color.

[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.

[HeightList](#)

Specifies the height for the drop-down window.

[HideDropDownButton](#)

Returns a value that determines whether the drop down button is hidden when the control loses the focus.

[HTMLPicture](#)

Adds or replaces a picture in HTML captions.

[hWnd](#)

Retrieves the control's window handle.

[hWndDropDown](#)

Retrieves a value that indicates the drop down window's

	handle.
Images	Sets a runtime the control's image list. The Handle should be a handle to an Image List control.
ImageSize	Retrieves or sets the size of icons the control displays.
Indent	Retrieves or sets the amount, in pixels, that child items are indented relative to their parent items.
IntegralHeight	Gets or sets a value indicating whether the control should resize to avoid showing partial items.
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.
Key	Replaces a virtual key.
LabelColumnIndex	Specifies a different column (index) to be displayed on the control's label, while the SingleEdit property is True.
LabelHeight	Retrieves or sets a value that indicates the label's height, in pixels.
LabelText	Specifies the HTML caption to be displayed in the control's label when SearchColumnIndex property points to a not-existing column, AdjustSearchColumn property in False, the SingleEdit property is True, and the Style property is DropDownList.
LinesAtRoot	Link items at the root of the hierarchy.
Locked	Determines whether a control can be edited.
MarkSearchColumn	Retrieves or sets a value that indicates whether the searching column is marked or unmarked
MinHeightList	Retrieves or sets a value that indicates the minimal height of dropped list control.
MinWidthList	Retrieves or sets a value that indicates the minimal width of dropped list control.
PutItems	Adds an array of integer, long, date, string, double, float, or variant arrays to the control.
RadiolImage	Retrieves or sets a value that indicates the image used by cells of radio type.
Refresh	Refreshes the control's content.

RemoveSelection	Removes the selected items (including the descendents)
Replacelcon	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.
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.
ScrollOnDrop	Specifies a value that indicates whether the drop down list is scrolled when it is shown.
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.
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.
Select	Finds and selects an item given data on the column.
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.
ShowClearButton	Shows or hides the control's clear-button.
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.
ShowLockedItems	Retrieves or sets a value that indicates whether locked/fixed items are visible or hidden.
ShowToolTip	Shows the specified tooltip at given position.
SingleEdit	Retrieves or sets a value that indicates whether the control displays a single edit control or multiple edit controls.
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.
Style	Retrieves or sets a value that indicates the control's style.
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.

[TreeColumnIndex](#)

Retrieves or sets a value indicating the column's index where the hierarchy will be displayed.

[UnboundHandler](#)

Specifies the control's unbound handler.

[UseMnemonic](#)

Gets or sets a value indicating whether the control interprets an ampersand character (&) in the item's caption to be an access key prefix character.

[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.

[Value](#)

Specifies the selected value for controls with a single column.

[Version](#)

Retrieves the control's version.

[VisualAppearance](#)

Retrieves the control's appearance.

[WidthList](#)

Specifies the width for drop-down window.

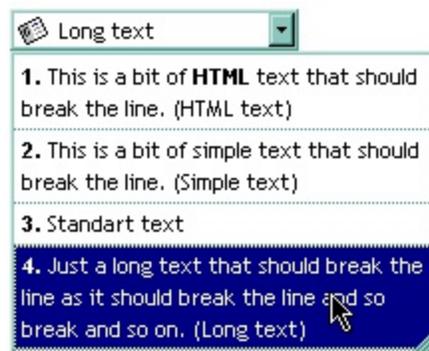
property ComboBox.AdjustSearchColumn as Boolean

Returns or sets a property that indicates whether the SearchColumnIndex could point to a hidden column.

Type	Description
Boolean	A boolean expression that indicates whether the control adjusts the SearchColumnIndex property to point to a visible column.

By default, the AdjustSearchColumn property is True. If the AdjustSearchColumn property is True, when the user hides a column using the [Visible](#) property, the [SearchColumnIndex](#) property gets a new value in order to point to a visible column. The SearchColumnIndex property indicates the column being displayed in the control's label area, when [SingleEdit](#) property is True. Use the AdjustSearchColumn property to let the control to display a hidden column in the control's label area.

For instance, lets say that we have a drop down window with some information and we need to display other information in the control's label when the user selects an item. The sample adds two columns, one is displayed in the drop down window, and the second column (hidden column) is displayed on the control's label.



The VB sample looks like follows:

```
With ComboBox1
    .BeginUpdate

    .AdjustSearchColumn = False
    .SearchColumnIndex = 1
    .HeaderVisible = False
    .SingleEdit = True
    .ColumnAutoResize = True
    .ScrollBySingleLine = True
```

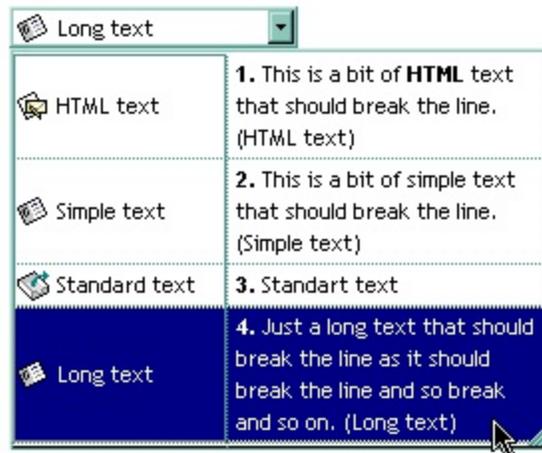
```
.AllowSizeGrip = True
.DrawGridLines = exAllLines
.Alignment = LeftAlignment
.WidthList = 216
```

```
With .Columns.Add("Visible")
    .Def(exCellSingleLine) = False
End With
With .Columns.Add("Hidden")
    .Visible = False
End With
```

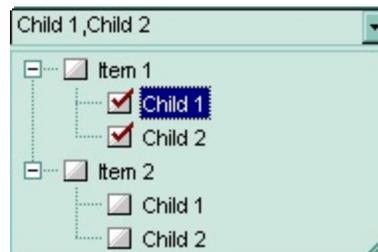
```
With .Items
    Dim h As HITEM
    h = .AddItem("1. This is a bit of HTML text that should break the line. (HTML text)")
    .CellCaptionFormat(h, 0) = exHTML
    .CellCaption(h, 1) = "HTML text"
    .CellImage(h, 1) = 1
    h = .AddItem("2. This is a bit of simple text that should break the line. (Simple text)")
    .CellCaptionFormat(h, 0) = exHTML
    .CellCaption(h, 1) = "Simple text"
    .CellImage(h, 1) = 2
    h = .AddItem("3. Standart text")
    .CellCaptionFormat(h, 0) = exHTML
    .CellCaption(h, 1) = "Standard text"
    .CellImage(h, 1) = 3
    h = .AddItem("4. Just a long text that should break the line as it should break the line and so break and so on. (Long text)")
    .CellCaptionFormat(h, 0) = exHTML
    .CellCaption(h, 1) = "Long text"
    .CellImage(h, 1) = 2
End With

.EndUpdate
End With
```

If we show the "Hidden" column like the sample looks like follows:



For instance you can use this feature if the [Style](#) property is DropDownList property to display an owner draw cell in the control's label area. Use the [CellOwnerDraw](#) property to draw a cell.



The following sample displays the list of items being checked separated by comma using the control's owner draw feature (you are simulating a check list editor):

```
Implements EXCOMBOBOXLibCtl.IOwnerDrawHandler
```

```
Option Explicit
```

```
Private Type RECT
```

```
    left As Long
```

```
    top As Long
```

```
    right As Long
```

```
    bottom As Long
```

```
End Type
```

```
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hDC As Long,  
    ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As  
    Long
```

```
Private Const DT_VCENTER = &H4;
```

```
Private Sub Form_Load()
```

With ComboBox1

```
.HeaderVisible = False  
.Style = DropDownList  
.SingleEdit = True  
.AllowSizeGrip = True  
.ColumnAutoResize = True  
.AdjustSearchColumn = False  
.LinesAtRoot = exLinesAtRoot  
.IntegralHeight = True  
.FullRowSelect = False
```

With .Columns

```
.Add "Column 1"  
With .Add("owner draw column")  
    .Visible = False  
    ComboBox1.SearchColumnIndex = .Index  
End With  
End With
```

With .Items

```
Dim h As HITEM, hC As HITEM, i As Long  
For i = 1 To 2  
    h = .AddItem("Item " & i)  
    .CellHasCheckBox(h, 0) = True  
    Set .CellOwnerDraw(h, 1) = Me  
    hC = .InsertItem(h, , "Child 1")  
    .CellHasCheckBox(hC, 0) = True  
    Set .CellOwnerDraw(hC, 1) = Me  
    hC = .InsertItem(h, , "Child 2")  
    .CellHasCheckBox(hC, 0) = True  
    Set .CellOwnerDraw(hC, 1) = Me  
    .ExpandItem(h) = True  
    If (i = 1) Then  
        .SelectItem(h) = True  
    End If  
Next
```

```

End With
End With
End Sub

Private Sub IOwnerDrawHandler_DrawCell(ByVal hDC As Long, ByVal left As Long, ByVal
top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As Long, ByVal
Column As Long, ByVal Source As Object)
    Dim rc As RECT
    With rc
        .left = left + 3
        .right = right
        .top = top
        .bottom = bottom
    End With
    Dim s As String
    s = ""
    With ComboBox1
        Dim i
        For Each i In .Items
            If (.Items.CellState(i, 0) = 1) Then
                s = s + If(Len(s) > 0, ",", "") + .Items.CellCaption(i, 0)
            End If
        Next
    End With
    DrawText hDC, s, Len(s), rc, DT_VCENTER
End Sub

```

The sample adds two columns. The hidden column is used to let user paint whatever he needs.

The following C++ sample adds two columns, and displays values from the hidden column when a new items is selected:

```

COleVariant vtMissing; V_VT( &vtMissing; ) = VT_ERROR;
m_combobox.BeginUpdate();
m_combobox.SetAdjustSearchColumn( FALSE );
m_combobox.SetSearchColumnIndex( 1 );
m_combobox.SetHeaderVisible( FALSE );

```

```

m_combobox.SetSingleEdit( TRUE );
m_combobox.SetColumnAutoResize( TRUE );
m_combobox.SetScrollBySingleLine( TRUE );
m_combobox.SetAllowSizeGrip( TRUE );
m_combobox.SetDrawGridLines( 1 );
m_combobox.SetAlignment( 0 );
m_combobox.SetWidthList( vtMissing, 216 );

CColumns columns = m_combobox.GetColumns();
CColumn column1( V_DISPATCH( &columns.Add;( "Visible" ) ) );
column1.SetDef( 16, COleVariant( VARIANT_FALSE ) );
CColumn column2( V_DISPATCH( &columns.Add;( "Hidden" ) ) );
column2.SetVisible( FALSE );

```

```

CItems items = m_combobox.GetItems();
COleVariant vtSColumn(long(1)), vtItem = items.AddItem( COleVariant( "1. This item is
shown in the drop down portion of the control. The second column displays information
in the control's label." ) );
items.SetCellCaption( vtItem, vtSColumn, COleVariant( "Item 1" ) );
vtItem = items.AddItem( COleVariant( "2. This part is never shown in the control's label
area." ) );
items.SetCellCaption( vtItem, vtSColumn, COleVariant( "Item 2" ) );
m_combobox.EndUpdate();

```

The following VB.NET sample adds two columns, and displays values from the hidden column when a new items is selected:

```

With AxComboBox1
    .BeginUpdate()
    .AdjustSearchColumn = False
    .SearchColumnIndex = 1
    .HeaderVisible = False
    .SingleEdit = True
    .ColumnAutoResize = True
    .ScrollBySingleLine = True
    .AllowSizeGrip = True
    .DrawGridLines = EXCOMBOBOXLib.GridLinesEnum.exAllLines
    .Alignment = EXCOMBOBOXLib.AlignmentEnum.LeftAlignment

```

```
.set_WidthList(216)
```

```
With .Columns.Add("Visible")
```

```
    .Def(EXCOMBOBOXLib.DefColumnEnum.exCellSingleLine) = False
```

```
    .Def(EXCOMBOBOXLib.DefColumnEnum.exCellCaptionFormat) =
```

```
EXCOMBOBOXLib.CaptionFormatEnum.exHTML
```

```
End With
```

```
With .Columns.Add("Hidden")
```

```
    .Visible = False
```

```
End With
```

```
With .Items
```

```
    Dim h As Integer = .AddItem("1. This is a bit of HTML text that should break the line.  
(HTML text)")
```

```
    .CellCaption(h, 1) = "HTML text"
```

```
    .CellImage(h, 1) = 1
```

```
    h = .AddItem("2. This is a bit of simple text that should break the line. (Simple text)")
```

```
    .CellCaption(h, 1) = "Simple text"
```

```
    .CellImage(h, 1) = 2
```

```
    h = .AddItem("3. Standart text")
```

```
    .CellCaption(h, 1) = "Standard text"
```

```
    .CellImage(h, 1) = 3
```

```
    h = .AddItem("4. Just a long text that should break the line as it should break the line  
and so break and so on. (Long text)")
```

```
    .CellCaption(h, 1) = "Long text"
```

```
    .CellImage(h, 1) = 2
```

```
End With
```

```
.EndUpdate()
```

```
End With
```

The following C# sample adds two columns, and displays values from the hidden column when a new items is selected:

```
axComboBox1.BeginUpdate();  
axComboBox1.AdjustSearchColumn = false;  
axComboBox1.SearchColumnIndex = 1;  
axComboBox1.HeaderVisible = false;  
axComboBox1.SingleEdit = true;
```

```
axComboBox1.ColumnAutoSize = true;
axComboBox1.ScrollBySingleLine = true;
axComboBox1.AllowSizeGrip = true;
axComboBox1.DrawGridLines = EXCOMBOBOXLib.GridLinesEnum.exAllLines;
axComboBox1.Alignment = EXCOMBOBOXLib.AlignmentEnum.LeftAlignment;
axComboBox1.set_WidthList(216);
```

```
EXCOMBOBOXLib.Columns columns = axComboBox1.Columns;
EXCOMBOBOXLib.Column column = columns.Add("Visible") as EXCOMBOBOXLib.Column;
column.set_Def(EXCOMBOBOXLib.DefColumnEnum.exCellSingleLine, false);
column.set_Def(EXCOMBOBOXLib.DefColumnEnum.exCellCaptionFormat,EXCOMBOBOXLib
```

```
column = columns.Add("Hidden") as EXCOMBOBOXLib.Column;
column.Visible = false;
EXCOMBOBOXLib.Items items = axComboBox1.Items;
```

```
int h = items.AddItem("1. This is a bit of HTML text that should break the line. (HTML
text)");
```

```
items.set_CellCaption(h, 1, "HTML text" );
```

```
items.set_CellImage(h, 1, 1);
```

```
h = items.AddItem("2. This is a bit of simple text that should break the line. (Simple text)");
```

```
items.set_CellCaption(h, 1,"Simple text");
```

```
items.set_CellImage(h, 1, 2);
```

```
h = items.AddItem("3. Standart text");
```

```
items.set_CellCaption(h, 1,"Standard text");
```

```
items.set_CellImage(h, 1, 3);
```

```
h = items.AddItem("4. Just a long text that should break the line as it should break the line
and so break and so on. (Long text)");
```

```
items.set_CellCaption(h, 1,"Long text");
```

```
items.set_CellImage(h, 1,2);
```

```
axComboBox1.EndUpdate();
```

The following VFP sample adds two columns, and displays values from the hidden column when a new items is selected:

```
With thisform.ComboBox1
    .BeginUpdate
        .AdjustSearchColumn = .f.
```

```
.SearchColumnIndex = 1
.HeaderVisible = .f
.SingleEdit = .t
.ColumnAutoResize = .f
.ScrollBySingleLine = .t
.AllowSizeGrip = .t
.DrawGridLines = 1 && exAllLines
.Alignment = 0 && LeftAlignment
.WidthList = 216
```

```
With .Columns.Add("Visible")
    .Def(16) = .f && exCellSingleLine
    .Def(17) = 1 && exHTML
```

```
EndWith
```

```
With .Columns.Add("Hidden")
```

```
    .Visible = .f
EndWith
```

```
With .Items
```

```
    .DefaultItem = .AddItem("1. This is a bit of HTML text that should break the line.
(HTML text)")
```

```
    .CellCaption(0, 1) = "HTML text"
```

```
    .CellImage(0, 1) = 1
```

```
    .DefaultItem = .AddItem("2. This is a bit of simple text that should break the line.
(Simple text)")
```

```
    .CellCaption(0, 1) = "Simple text"
```

```
    .CellImage(0, 1) = 2
```

```
    .DefaultItem = .AddItem("3. Standart text")
```

```
    .CellCaption(0, 1) = "Standard text"
```

```
    .CellImage(0, 1) = 3
```

```
    h = .AddItem("4. Just a long text that should break the line as it should break the
line and so break and so on. (Long text)")
```

```
    .CellCaption(0, 1) = "Long text"
```

```
    .CellImage(0, 1) = 2
```

```
EndWith
```

```
.EndUpdate
```

```
EndWith
```

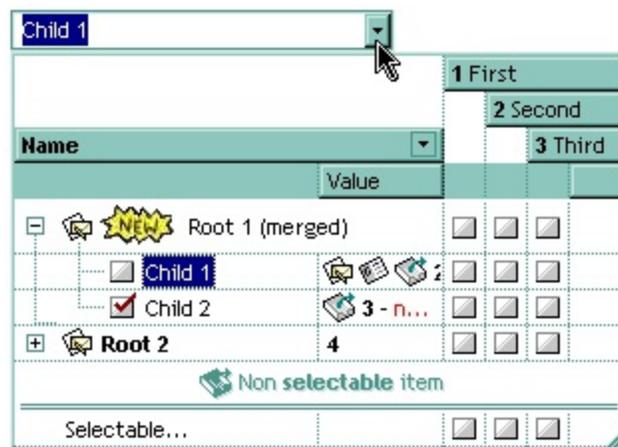

property ComboBox.Alignment as AlignmentEnum

Retrieves or sets a value that indicates the drop down list's alignment.

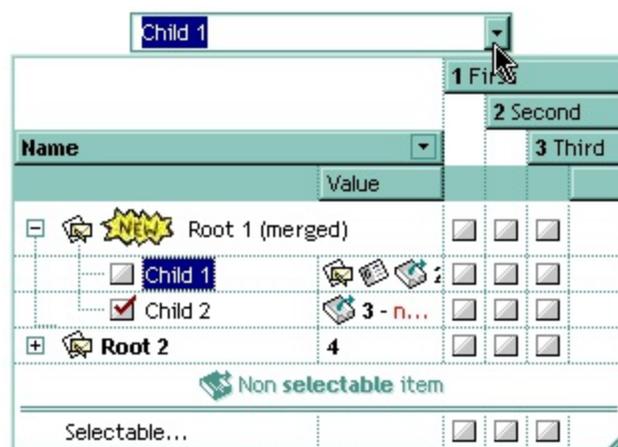
Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the drop down list's alignment.

By default, the Alignment property is set to RightAlignment. Use the Alignment property to change the alignment of the drop down list. The property has effect only if the control's [Style](#) is DropDown or DropDownList. The property has effect only if the client area width is less than [MinWidthList](#) property. Use the [Alignment](#) property to align the cells in the column. Use the [HeaderAlignment](#) property to specify the alignment of the column's caption in the header. Use the [CellAlignment](#) property to align the caption in the cell.

The following screen shot shows a drop down control when Alignment property is LeftAlignment:



The following screen shot shows a drop down control when Alignment property is CenterAlignment:



The following screen shot shows a drop down control when Alignment property is

RightAlignment:

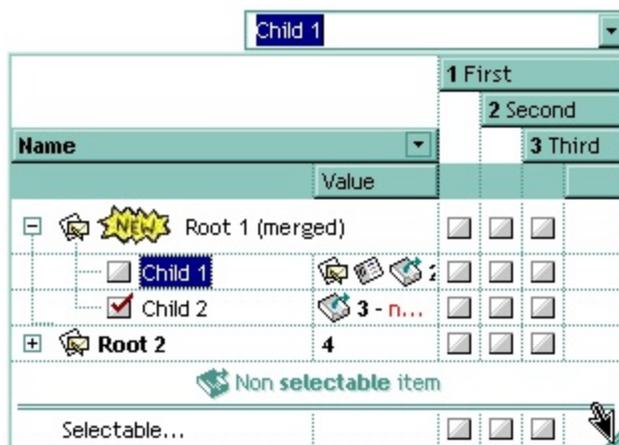
Child 1		1 First	2 Second	3 Third
Name	Value			
<input type="checkbox"/>  Root 1 (merged)		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Child 1	 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Child 2	 3 - n...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>  Root 2	4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 Non selectable item				
Selectable...		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

property ComboBox.AllowHResize as Boolean

Specifies whether the user can resize the width of the drop down portion when AllowSizeGrip is True.

Type	Description
Boolean	A boolean expression that indicates whether the user is able to resize the width of drop-down portion while dragging the size grip.

Use the AllowHResize, [AllowVResize](#) properties to specify whether the width, height of the drop down portion of the control are resizable. Use the [AllowSizeGrip](#) property to specify whether the drop down portion of the control is resizable at runtime. Use the [WidthList](#) property to specify the width of the drop down portion of the control. Use the [HeightList](#) property to specify the height of the drop down portion of the control. Use the [IntegralHeight](#) property to avoid showing partial items.

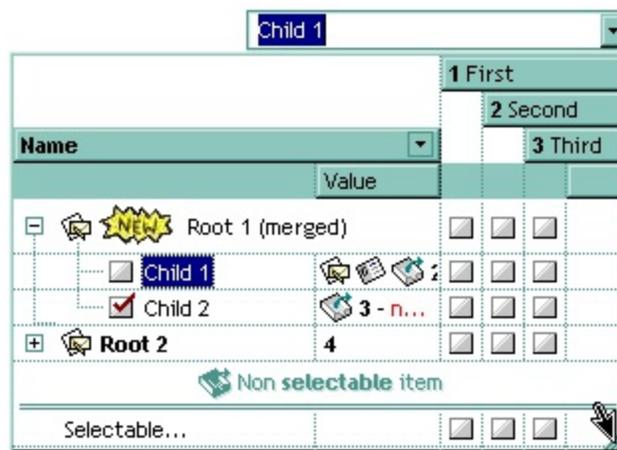


property ComboBox.AllowSizeGrip as Boolean

Retrieves or sets a value indicating whether the drop-down window is resizable at runtime.

Type	Description
Boolean	A boolean expression indicating whether the drop-down window is resizable at runtime.

By default, the AllowSizeGrip property is False. If the AllowSizeGrip is True, the control displays a size grip on the bottom-right side of the list window, and it can be used to resize the list window at runtime. Use the [MinWidthList](#) and [MinHeightList](#) properties to specify the minimum size of the drop down window. Use the [AllowHResize](#), [AllowVResize](#) properties to specify whether the width, height of the drop down portion of the control are resizable.

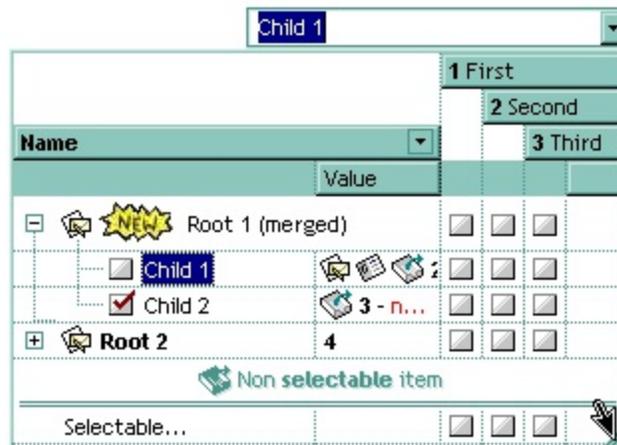


property ComboBox.AllowVResize as Boolean

Specifies whether the user can resize the height of the drop down portion when AllowSizeGrip is True.

Type	Description
Boolean	A boolean expression that indicates whether the user can resize the height of the drop down portion by dragging the size grip.

Use the [AllowHResize](#), AllowVResize properties to specify whether the width, height of the drop down portion of the control are resizable. Use the [AllowSizeGrip](#) property to specify whether the drop down portion of the control is resizable at runtime. Use the [WidthList](#) property to specify the width of the drop down portion of the control. Use the [HeightList](#) property to specify the height of the drop down portion of the control. Use the [IntegralHeight](#) property to avoid showing partial items.



property ComboBox.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 ComboBox1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        .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 AxComboBox1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent) Handles AxComboBox1.MouseMoveEvent
    With AxComboBox1
        .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 axComboBox1_MouseMoveEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent e)
{
    axComboBox1.ShowToolTip(axComboBox1.get_AnchorFromPoint(-1, -1));
}
```

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

```
void OnMouseMoveCombobox1(short Button, short Shift, long X, long Y)
{
    COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
    m_comboBox.ShowToolTip( m_comboBox.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 .ComboBox1
        .ShowToolTip(.AnchorFromPoint(-1, -1))
    EndWith
endwith
```

property ComboBox.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. <i>The Client object in the skin, defines the client area of the control's label. The edit control or the drop down button are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The normal.ebn file contains such of objects. Use the eXButton's Skin builder to view or change this file.</i>

Use the Appearance property to change the appearance of the control. Use the [DropDownBorder](#) property to change the border of the drop down portion of the control. Use the [HeaderAppearance](#) property to change the control's header bar appearance. Use the [Alignment](#) property to specify the alignment of the drop down portion of the control. Use the [Style](#) property to specify the control's style. Use the [Add](#) method to add new skins to the control. Use the [BackColorEdit](#) property to specify the background color of the control's edit box.

The following picture, shows the frame of the control before running the sample, and after running the sample, using the [skin](#):



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

```
With ComboBox1
    .BeginUpdate
        .VisualAppearance.Add &H16, "c:\temp\normal.ebn"
        .Appearance = &H16000000
        .BackColorEdit = RGB(250, 250, 250)
    .EndUpdate
End With
```

```
.EndUpdate
```

```
End With
```

The following VB.NET sample changes the visual aspect of the borders of the control:

```
With AxComboBox1
```

```
    .BeginUpdate()
```

```
    .VisualAppearance.Add(&H16, "c:\temp\normal.ebn")
```

```
    .Appearance = &H16000000
```

```
    .BackColorEdit = Color.FromArgb(250, 250, 250)
```

```
    .EndUpdate()
```

```
End With
```

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

```
axComboBox1.BeginUpdate();
```

```
axComboBox1.VisualAppearance.Add(0x16, "c:\\temp\\normal.ebn");
```

```
axComboBox1.Appearance = (EXCOMBOBOXLib.AppearanceEnum)0x16000000;
```

```
axComboBox1.BackColorEdit = Color.FromArgb(250, 250, 250);
```

```
axComboBox1.EndUpdate();
```

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

```
m_comboBox.BeginUpdate();
```

```
m_comboBox.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\normal.ebn" ) );
```

```
m_comboBox.SetAppearance( 0x16000000 );
```

```
m_comboBox.SetBackColorEdit( RGB(250,250,250) );
```

```
m_comboBox.EndUpdate();
```

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

```
with thisform.ComboBox1
```

```
    .BeginUpdate
```

```
        .VisualAppearance.Add(0x16, "c:\temp\normal.ebn")
```

```
        .Appearance = 0x16000000
```

```
        .BackColorEdit = RGB(250, 250, 250)
```

```
    .EndUpdate
```

```
endwith
```


method `ComboBox.ApplyFilter ()`

Applies the filter. */*not supported in the lite version*/*

Type

Description

The `ApplyFilter` method updates the control's content once that user sets the filter using the [Filter](#) and [FilterType](#) properties. Use the [VisibleItemCount](#) property to specify the number of visible items in the list. 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 [FilterBarCaption](#) property to specify a new caption for the control's filter bar when a filter is applied. Use the [FilterBarHeight](#) property to specify the height of the control's filter bar. Use the [FilterInclude](#) property to specify whether the child items should be included to the list when the user applies the filter. Use the [FilterCriteria](#) property to filter items using AND, OR or NOT operators between columns.

The following VB sample filters for items that contains the string you typed so far (the control's [Style](#) property is `DropDownList`):

```
Option Explicit
```

```
Dim s As String
```

```
Private Sub ComboBox1_FilterChange()
```

```
    With ComboBox1.Columns(0)
```

```
        If (.FilterType = exAll) Then
```

```
            s = ""
```

```
        End If
```

```
    End With
```

```
    If (Len(s) > 0) Then
```

```
        ComboBox1.FilterBarCaption = "Include only the items that contain ' " & s & "' . "
```

```
    End If
```

```
End Sub
```

```
Private Sub ComboBox1_KeyDown(KeyCode As Integer, Shift As Integer)
```

```
    If (KeyCode = vbKeyDelete) Then
```

```
        s = ""
```

```
        ComboBox1.ClearFilter
```

```
    End If
```

```
End Sub
```

```
Private Sub ComboBox1_KeyPress(KeyAscii As Integer)
```

```
    If (KeyAscii = vbKeyBack) Then
```

```
        If (Len(s) > 0) Then
```

```
            s = Left(s, Len(s) - 1)
```

```
        End If
```

```
    Else
```

```
        s = s + Chr(KeyAscii)
```

```
    End If
```

```
With ComboBox1
```

```
    If (Len(s) > 0) Then
```

```
        With .Columns(0)
```

```
            .Filter = "*" & s & "*"
```

```
            .FilterType = exPattern
```

```
        End With
```

```
        .ApplyFilter
```

```
    Else
```

```
        .ClearFilter
```

```
    End If
```

```
End With
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    ' The data is loaded at design time, using the
```

```
    ' control's Template page
```

```
With ComboBox1.Columns(0)
```

```
    .AutoSearch = exContains
```

```
End With
```

```
End Sub
```

The following VB sample filters "Child 1" and "Child 2" items:

```
With ComboBox1
```

```
    With .Columns(0)
```

```
        .Filter = "Child 1|Child 2"
```

```
        .FilterType = exPattern
```

```
    End With
```

```
.ApplyFilter
```

```
End With
```

The following C++ sample filters "Child 1" and "Child 2" items:

```
CColumn column = m_combobox.GetColumns().GetItem(COleVariant(long(0)));  
column.SetFilter( "Child 1|Child 2" );  
column.SetFilterType( 3 /*exPattern*/ );  
m_combobox.ApplyFilter();
```

The following VB.NET sample filters "Child 1" and "Child 2" items:

```
With AxComboBox1  
  With .Columns(0)  
    .Filter = "Child 1|Child 2"  
    .FilterType = EXCOMBOBOXLib.FilterTypeEnum.exPattern  
  End With  
  .ApplyFilter()  
End With
```

The following C# sample filters "Child 1" and "Child 2" items:

```
axComboBox1.Columns[0].Filter = "Child 1|Child 2";  
axComboBox1.Columns[0].FilterType = EXCOMBOBOXLib.FilterTypeEnum.exPattern;  
axComboBox1.ApplyFilter();
```

The following VFP sample filters "Child 1" and "Child 2" items:

```
With thisform.ComboBox1  
  With .Columns.Item(0)  
    .Filter = "Child 1|Child 2"  
    .FilterType = 3 && exPattern  
  EndWith  
  .ApplyFilter  
EndWith
```


property ComboBox.ASCIIUpper as String

Specifies the set of upper characters.

Type	Description
String	A string expression that indicates the set of upper characters used by auto search feature.

The [ASCIILower](#) and ASCIIUpper properties helps you to specify the set of characters that are used by the auto search feature. If you want to make the auto search feature case sensitive you have to use ASCIIUpper = "" . By default, the ASCIIUpper property is = "ABCDEFGHIJKLMNOPQRSTUVWXYZÜÉÂĂŔŁÇĘĚĎÎĚÔÖŇÚŮÁÍÓÚŇ"

property ComboBox.AssignEditImageOnSelect(ColIndex as Long) as Boolean

Assigns the edit's icon when the selection is changed.

Type	Description
ColIndex as Long	A long expression that indicates the index of column.
Boolean	A boolean expression that indicates whether the control changes the edit's icon when selection is changed.

By default, the cell's icon is not displayed in the control's label. The `AssignEditImageOnSelect` property specifies whether the label displays the cell's icon when selection is changed. The [EditImage](#) property assigns an icon to the column's edit box, if the [Style](#) property is `DropDown`. Use the [CellImage](#) property to assign an icon to a cell. Use the [Images](#) method to assign a list of icons to the control. The control fires the [SelectionChanged](#) event when the user selects a new item.

method **ComboBox.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 ComboBox1_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 ComboBox.AutoComplete as Boolean

Retrieves or sets a value that indicates whether auto complete feature is enabled or disabled.

Type	Description
Boolean	A boolean expression that indicates whether the control completes automatically all the edit controls when the selected item is changed.

Use the AutoComplete property to disable auto complete feature. If the AutoComplete property is True, each edit of the control contains the data of the selected item in the control's list. To get the caption for a specified edit control uses the [EditText](#) property of the control. The [NotInList](#) event may **not** be fired if the [AutoComplete](#) property is True. Use the [AutoSearch](#) property to enable incremental searching feature. The [AutoSelect](#) property retrieves or sets a value indicating whether the control selects the portion of text that doesn't match with the selected item. The control updates the control's label (updates each edit control in the label area), when the selection is changed based on the selected item. For instance, if you click an item, the control automatically updates the control's label with the values in the selected item. The control fires the [SelectionChanged](#) event when the user selects a new items.

property ComboBox.AutoComplete as AutoDragEnum

Gets or sets a value that indicates the way the component supports the AutoDrag feature.

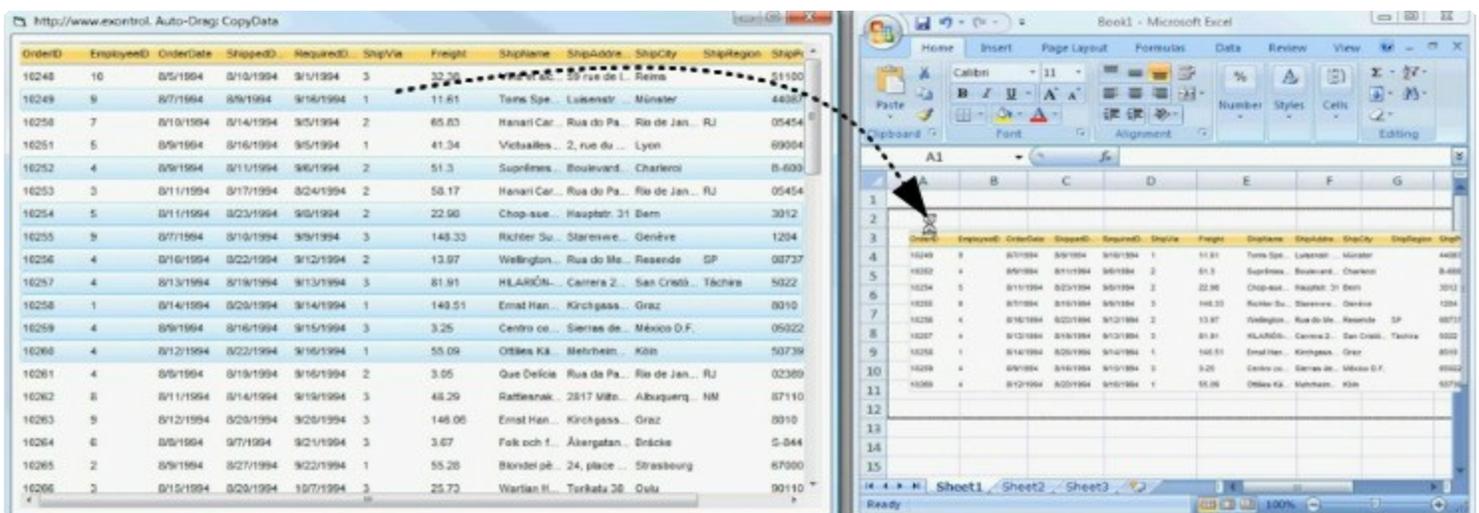
Type

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. Also, you can use the AutoDrag property (exAutoDragScroll + exAutoDragScrollOnShortTouch, or 4112) to let user scrolls the control's content when user touches a capacitive screen.



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
- exAutoDragPositionKeepIndent
- exAutoDragPositionAny

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 ComboBox.AutoDropDown as Boolean

Retrieves or sets a value that indicates whether the control's list automatically drops down once the user starts type into it.

Type	Description
Boolean	A boolean expression that indicates whether the control's list automatically drops down once the user starts type into it.

By default, the AutoDropDown property is False. The property has effect only if the control's [Style](#) is DropDown or DropDownList.

- If the Style property is DropDownList and AutoDropDown property is True, the control shows the drop down portion of the control when user clicks the control's label area, if the control's list is not empty. Use the [Add](#) method to add new columns. Use the [AddItem](#) or [InsertItem](#) method to add new items to the control's list.
- If the Style property is DropDown and the AutoDropDown property is True, the control shows the drop down portion as soon as the user starts type characters.
- If the Style property is Simple, the AutoDropDown property has no effect.

If the AutoDropDown property is False, the user can open the control's list by pressing F4 or by clicking the right drop down button. Use the [DropDown](#) method to show programmatically the drop down portion of the control.

If the AutoDropDown property is True, the user can automatically drop down portion of the control if any of the following key is pressed:

- digit keys from 0 to 9
- alpha keys from A to Z
- any of : ; + , - . ? ~ [{ \] } ' " ' "
- numeric keypad keys.

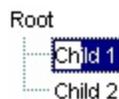
property ComboBox.AutoSearch as Boolean

Retrieves or sets a value that indicates whether auto-search feature is enabled or disabled.

Type	Description
Boolean	A boolean expression that indicates whether the auto-search feature is enabled or disabled.

By default, the AutoSearch property is True. The auto-search feature 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 [AutoComplete](#) property to enable or disable the auto complete feature. Use the [MarkSearchColumn](#) property to specify whether the control draws a rectangle around the searching column. Use the [EditText](#) property to get the text in the control's label area.

The control highlights the characters as the user types them:



property ComboBox.AutoComplete as Boolean

Retrieves or sets a value indicating whether the control selects the portion of text that doesn't match with the selected item

Type	Description
Boolean	A boolean expression indicating whether the control selects the portion of text that doesn't match with the selected item.

The AutoSelect property has effect only if the [AutoComplete](#) property is True. Use AutoSelect property to let the user types whatever they want into the control edit boxes without auto-completing the entire text of the focused edit box.

property ComboBox.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.

The ExComboBox 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. To change the background color of the control's locked area use [BackColorLock](#) property. Use the [BackColorEdit](#) property to specify the background color of the control's edit box. Use the [CellBackColor](#) property to specify the cell's background color. Use the [ItemBackColor](#) property to specify the item's background color. Use the [HeaderBackColor](#) and [HeaderForeColor](#) properties to defines the colors for the control's header bar.

property ComboBox.BackColorAlternate as Color

Specifies the background color used to display alternate items in the control.

Type	Description
Color	A color expression that indicates the alternate background color.

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 [CellBackColor](#) property to specify the cell's background color. Use the [ItemBackColor](#) property to specify the item's background color.

property ComboBox.BackColorEdit as Color

Specifies the control's edit background color.

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

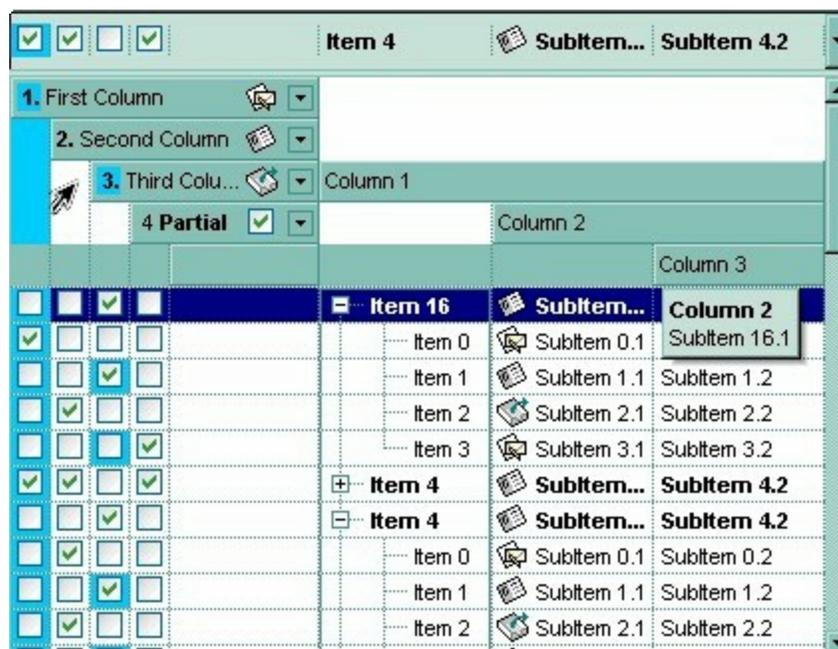
Use the BackColorEdit property to specify the background color of the control's edit box. Use the [BackColor](#) property to specify the background color of the control's drop down portion. Use the [CellBackColor](#) property to specify the cell's background color. Use the [ItemBackColor](#) property to specify the item's background color. The BackColorEdit property has effect only, if the [Style](#) property is not DropDownList. If the control is locked, the control panel specifies the background color for read only edit controls.

property ComboBox.BackColorLevelHeader as Color

Specifies the multiple levels header's background color. */*not supported in the lite version*/*

Type	Description
Color	A color expression that indicates 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. Use the [HeaderBackColor](#) and [HeaderForeColor](#) properties to define colors used to paint 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.

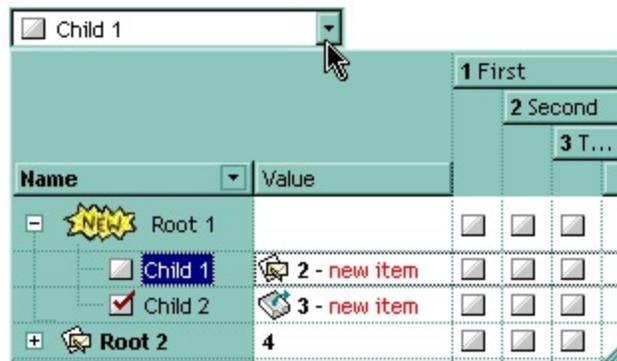


property ComboBox.BackColorLock as Color

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 ExComboBox 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. To change the foreground color of the control's locked area use ForeColorLock property. Use the [LockedItemCount](#) property to specify the count of locked items.



property ComboBox.BackColorSortBar as Color

Retrieves or sets a value that indicates the sort bar's background color. */*not supported in the lite version*/*

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 [HeaderBackColor](#) 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 ComboBox.BackColorSortBarCaption as Color

Returns or sets a value that indicates the caption's background color in the control's sort bar. */*not supported in the lite version*/*

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 ComboBox.Background(Part as BackgroundPartEnum) as Color

Returns or sets a value that indicates the background color for parts in the control. */*not supported in the lite version*/*

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.



The following VB sample changes the visual appearance for the drop down button. The sample uses the "☑" skin when the button is up, and uses the "☑" skin when the drop down button is down.

```
With ComboBox1
    .VisualAppearance.Add 4, "D:\Temp\ExComboBox.Help\fbardd.ebn"
    .VisualAppearance.Add 5, "D:\Temp\ExComboBox.Help\fbarddd.ebn"
    .Background(exDropDownButtonUp) = &H4000000
    .Background(exDropDownButtonDown) = &H5000000
End With
```

The following C++ sample changes the visual appearance for the drop down button:

```
#include "appearance.h"
m_combobox.GetVisualAppearance().Add( 0x12, COleVariant(
```

```
"D:\\Temp\\ExComboBox.Help\\fbardd.ebn" );  
m_combobox.GetVisualAppearance().Add( 0x13, COleVariant(  
"D:\\Temp\\ExComboBox.Help\\fbarddd.ebn" ) );  
m_combobox.SetBackground( 4 /*exDropDownButtonUp*/, 0x12000000 );  
m_combobox.SetBackground( 5 /*exDropDownButtonDown*/, 0x13000000 );
```

The following VB.NET sample changes the visual appearance for the drop down button:

```
With AxComboBox1  
    .VisualAppearance.Add(4, "D:\\Temp\\ExComboBox.Help\\fbardd.ebn")  
    .VisualAppearance.Add(5, "D:\\Temp\\ExComboBox.Help\\fbarddd.ebn")  
    .set_Background(EXCOMBOBOXLib.BackgroundPartEnum.exDropDownButtonUp,  
&H4000000)  
    .set_Background(EXCOMBOBOXLib.BackgroundPartEnum.exDropDownButtonDown,  
&H5000000)  
End With
```

The following C# sample changes the visual appearance for the drop down button:

```
axComboBox1.VisualAppearance.Add(4, "D:\\Temp\\ExComboBox.Help\\fbardd.ebn");  
axComboBox1.VisualAppearance.Add(5, "D:\\Temp\\ExComboBox.Help\\fbarddd.ebn");  
axComboBox1.set_Background(EXCOMBOBOXLib.BackgroundPartEnum.exDropDownButto  
0x4000000);  
axComboBox1.set_Background(EXCOMBOBOXLib.BackgroundPartEnum.exDropDownButto  
0x5000000);
```

The following VFP sample changes the visual appearance for the drop down button:

```
with thisform.ComboBox1  
    .VisualAppearance.Add(14, "D:\\Temp\\ExComboBox.Help\\fbardd.ebn")  
    .VisualAppearance.Add(15, "D:\\Temp\\ExComboBox.Help\\fbarddd.ebn")  
    .Background(4) = 234881024    && exDropDownButtonUp  
    .Background(5) = 251658240    && exDropDownButtonDown  
endwith
```

method **ComboBox.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.

The following VB sample uses the BeginUpdate and EndUpdate methods while loading columns 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

ComboBox1.BeginUpdate
For Each f In rs.Fields
    ComboBox1.Columns.Add f.Name
Next
ComboBox1.PutItems rs.GetRows()
ComboBox1.EndUpdate
```

The following VB sample prevents painting the control during loading columns and items:

```
With ComboBox1
    .BeginUpdate
    .Columns.Add ("Column 1")
    With .Items
        .AddItem "Item 1"
        .AddItem "Item 2"
    End With
    .EndUpdate
End With
```

The following C++ sample prevents painting the control during loading columns and items:

```
m_combobox.BeginUpdate();
CColumns columns = m_combobox.GetColumns();
```

```
columns.Add( "Column 1" );  
CItems items = m_combobox.GetItems();  
items.AddItem( COleVariant("Item 1") );  
items.AddItem( COleVariant("Item 2") );  
items.AddItem( COleVariant("Item 3") );  
m_combobox.EndUpdate();
```

The following VB.NET sample prevents painting the control during loading columns and items:

```
With AxComboBox1  
    .BeginUpdate()  
    .Columns.Add("Column 1")  
    With .Items  
        .AddItem("Item 1")  
        .AddItem("Item 2")  
    End With  
    .EndUpdate()  
End With
```

The following C# sample prevents painting the control during loading columns and items:

```
axComboBox1.BeginUpdate();  
axComboBox1.Columns.Add("Column 1");  
axComboBox1.Items.AddItem("Item 1");  
axComboBox1.Items.AddItem("Item 2");  
axComboBox1.EndUpdate();
```

The following VFP sample prevents painting the control during loading columns and items:

```
With thisform.ComboBox1  
    .BeginUpdate  
    .Columns.Add ("Column 1")  
    With .Items  
        .AddItem("Item 1")  
        .AddItem("Item 2")  
    EndWith  
    .EndUpdate  
EndWith
```


property ComboBox.CheckImage(State as CheckStateEnum) as Long

Retrieves or sets a value that indicates the image used by cells of checkbox type.

Type	Description
State as CheckStateEnum	A CheckStateEnum expression that indicates the check's state: 0 means unchecked, 1 means checked, and 2 means partial checked.
Long	A long expression that indicates the index of image used to paint the cells of check box types. 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 [RadiolImage](#) properties to define icons for radio and check box cells. Use the [CellHasCheckBox](#) property to assign a check box to a cell. Use the [CellHasRadioButton](#) property to assign a radio button to a cell. The control fires the [CellStateChanged](#) event when user changes the cell's checkbox state (or radio button as well). Use the [Images](#) method to assign a list of icons to the control, at runtime. Use the [Replacelcon](#) property to update the control's list of icons at runtime. Use the [PartialCheck](#) property to allow partial check feature within the column.

method `ComboBox.ClearFilter ()`

Clears the filter. */*not supported in the lite version*/*

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` methodid is invoked. Use the [FilterBarCaption](#) property to change the caption of the control's filter bar. Use the [FilterBarHeight](#) property to hide the control's filter bar. Use the [FilterInclude](#) property to specify whether the child items should be included to the list when the user applies the filter.

property ComboBox.CloseOnClick as Boolean

Specifies whether the user closes the drop down portion of the control, by single click.

Type	Description
Boolean	A Boolean expression that indicates whether the user closes the drop down portion of the control using single click.

By default, the CloseOnClick property is True. Use the CloseOnClick property to disable closing the drop down portion of the control when user clicks an item. Use the [CloseOnDbClick](#) property to specify whether the user closes the drop down portion of the control when user double clicks an item in the control. The control fires the [SelectionChanged](#) event when the user clicks a new item. The control fires the [DbClick](#) event when user double clicks an item. The CloseOnClick property has effect only if the [Style](#) property is DropDown or DropDownList. The CloseOnDbClick property is set on False, if the CloseOnClick property is set on True.

property ComboBox.CloseOnDbIClk as Boolean

Specifies whether the user closes the drop down window by dbl click.

Type	Description
Boolean	A boolean expression that indicates whether the user closes the drop down window by dbl click or by simple click.

By default, the CloseOnDbIClk property is False. Use the CloseOnDbIClk property to specify whether the user closes the drop down portion of the control by double clicks an item. Use the [CloseOnClick](#) property to specify whether the user closes the drop down portion of the control when an item is clicked. The control fires the [SelectionChanged](#) event when the user clicks a new item. The control fires the [DbIClick](#) event when user double clicks an item. The CloseOnDbIClk property has effect only if the [Style](#) property is DropDown or DropDownList. The CloseOnClick property is set on False, if the CloseOnDbIClk property is set on True.

property `ComboBox.ColumnAutoSize` as Boolean

Returns or sets a value indicating whether the control will automatically size its visible columns to fit on the control's client area.

Type	Description
Boolean	A boolean expression indicating whether the control will automatically size its visible columns to fit on the control's client area.

By default, the `ColumnAutoSize` property is `False`. Use the `ColumnAutoSize` property to fit all visible columns in the control's client area. Use the [Add](#) method to add new columns to the control's [Columns](#) collection. Use the [Width](#) property to change the column's width. Use the [Visible](#) property to hide a column. If the `ColumnAutoSize` property is `True`, the control does not display the control's horizontal scroll bar. By default, the control adds scroll bars when required

property ComboBox.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 ItemFromPoint property determines the handle of the item from the cursor.** Use the [ItemFromPoint](#) property to get the item from point. 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 ComboBox1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        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
```

End Sub

The following C++ sample prints the caption of the column from the point:

```
#include "Columns.h"
#include "Column.h"
void OnMouseMoveComboBox1(short Button, short Shift, long X, long Y)
{
    long nColIndex = m_combobox.GetColumnFromPoint( X, Y );
    if ( nColIndex >= 0 )
    {
        CColumn column = m_combobox.GetColumns().GetItem( COleVariant( nColIndex ) );
        OutputDebugString( column.GetCaption() );
    }
}
```

The following VB.NET sample prints the caption of the column from the point:

```
Private Sub AxComboBox1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent) Handles
AxComboBox1.MouseMoveEvent
    With AxComboBox1
        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 axComboBox1_MouseMoveEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent e)
{
    int i = axComboBox1.get_ColumnFromPoint( e.x,e.y );
    if ( i >= 0 )
        System.Diagnostics.Debug.WriteLine( axComboBox1.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.ComboBox1  
    i = .ColumnFromPoint( x, y )  
    if ( i >= 0 )  
        wait window nowait .Columns(i).Caption  
    endif  
endwith
```

property ComboBox.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 to control columns. Use the Columns collection you can add, remove or change the control 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 [AddItem](#), [InsertItem](#), 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.

The following VB sample adds a column and some items to a drop down list control:

```
With ComboBox1
    .BeginUpdate
    .ColumnAutoSize = True
    .Columns.Add "Column 1"
    .PutItems Array(1, "Item 2", 3, 4, 5)
    .Items.SelectItem(.Items.FindItem("Item 2")) = True
    .EndUpdate
End With
```

The following C++ sample adds a column and some items to a drop down list control:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
m_combobox.BeginUpdate();
m_combobox.SetColumnAutoSize( TRUE );
CColumns columns = m_combobox.GetColumns();
columns.Add( "Column 1" );
CItems items = m_combobox.GetItems();
items.AddItem( COleVariant((long)1) );
items.AddItem( COleVariant("Item 2") );
items.AddItem( COleVariant((long)3) );
items.AddItem( COleVariant((long)4) );
items.AddItem( COleVariant((long)5) );
items.SetSelectedItem( items.GetFindItem(COleVariant("Item 2"), COleVariant(long(0))),
```

```
vtMissing ), TRUE );  
m_combobox.EndUpdate();
```

The following VB.NET sample adds a column and some items to a drop down list control:

```
With AxComboBox1  
    .BeginUpdate()  
    .ColumnAutoResize = True  
    .Columns.Add("Column 1")  
    Dim o() As Object = {1, "Item 2", 3, 4, 5}  
    .PutItems(o)  
    .Items.SelectItem(.Items.FindItem("Item 2")) = True  
    .EndUpdate()  
End With
```

The following C# sample adds a column and some items to a drop down list control:

```
axComboBox1.BeginUpdate();  
axComboBox1.ColumnAutoResize = true;  
axComboBox1.Columns.Add("Column 1");  
axComboBox1.Items.Add(1);  
axComboBox1.Items.Add("Item 2");  
axComboBox1.Items.Add(3);  
axComboBox1.Items.Add(4);  
axComboBox1.Items.Add(5);  
axComboBox1.Items.set_SelectItem(axComboBox1.Items.get_FindItem("Item 2",null,null),  
true);  
axComboBox1.EndUpdate();
```

The following VFP sample adds a column and some items to a drop down list control:

```
With thisform.ComboBox1  
    .BeginUpdate  
    .ColumnAutoResize = .t.  
    .Columns.Add ("Column 1")  
    With .Items  
        .AddItem(1)  
        .AddItem("Item 2")  
        .AddItem(3)
```

.AddItem(4)

.AddItem(5)

EndWith

.EndUpdate

EndWith

property ComboBox.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 ComboBox.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 [CellCaption](#) 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.

method `ComboBox.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. Use the [ExpandItem](#) property to expand or collapse an item. 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 ComboBox1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyC) And Shift = 2 Then
        Clipboard.Clear
        ComboBox1.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 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 AxComboBox1
    .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):

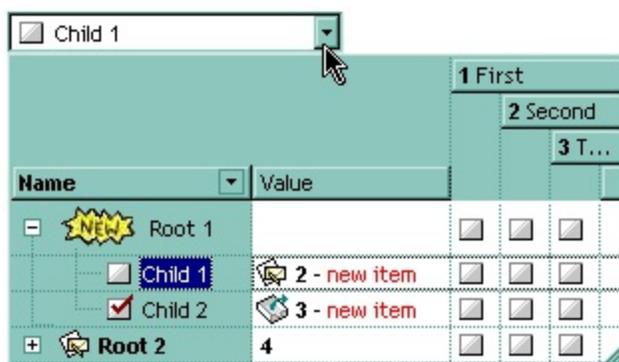
```
Clipboard.Clear;
axComboBox1.Copy();
```

property `ComboBox.CountLockedColumns` as Long

Retrieves or sets a value indicating the number of locked columns.

Type	Description
Long	A long expression that counts the locked columns.

The 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. To change the background color of the control's locked area use [BackColorLock](#) property. Use the [LockedItemCount](#) property to add or remove items fixed/locked to the top or bottom side of the control.



property ComboBox.DataSource as Object

Retrieves or sets a value that indicates the default item height.

Type	Description
Object	An ADO, DAO recordset being loaded.

Use the DataSource property to bind the control to an ADO recordset. Use PutItems property to fill the control from an array. When the user has changed the DataSource property, the control clears the old columns and items, and fetches the recordset columns and items. While control adding columns, the control fires [AddColumn](#) event While control loads the items from a recordset the control fires the [InsertItem](#) event. Use the [CellCaption](#) 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.

The following VB sample binds an ADO recordset to the ExComboBox 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 ComboBox1.DataSource = rs
```

The DataSource clears the columns collection, and fill the record set into the combobox, like a list. Use [SetParent](#) method to make your list a hierarchy.

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_combobox.BeginUpdate();
        m_combobox.SetColumnAutoResize( FALSE );
        m_combobox.SetDataSource( spRecordset );
        m_combobox.EndUpdate();
    }
}
catch ( _com_error& e )
{
    AfxMessageBox( e.Description() );
}
}

```

The #import statement imports definitions for ADO DB type library, that's used to fill the control

property ComboBox.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 height.

The `DefaultItemHeight` property specifies the default item height in pixels. The `DefaultItemHeight` property does not affect the items already added. By default, the `DefaultItemHeight` property is 17 pixels. 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. The [LabelHeight](#) property defines the height of the control's label in pixels.

property ComboBox.Description(Type as DescriptionTypeEnum) as String

Changes descriptions for different parts in the control.

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. The [FilterForVisible](#) property specifies whether the drop down portion of the control displays all the time a text-box editor to allow filtering items of the control.

property ComboBox.DrawGridLines as GridLinesEnum

Retrieves or sets a value that indicates whether the grid lines are visible or hidden.

Type	Description
GridLinesEnum	A boolean expression that indicates whether the grid lines are visible or hidden.

Use DrawGridLine property to show grid lines into the list items. Use the [LinesAtRoot](#) property specifies whether the control links the root items of the control. Use the [HasLines](#) property to specify whether the control draws the link between child items to their corresponding parent item.

property ComboBox.DropDown([Reserved as Variant]) as Boolean

Specifies whether the drop down portion of the control is visible or hidden.

Type	Description
Reserved as Variant	Reserved for future use only.
Boolean	A boolean expression that indicates whether the drop-down portion of the control is visible or hidden.

The DropDown property shows or hides the drop down portion of the control. Use the [hWndDropDown](#) property to retrieve the handle of the drop down window. The DropDown property has effect only if the control's [Style](#) property is DropDown or DropDownList. The control fires the [DropDown](#) event when the drop-down portion is shown. The control fires [DropUp](#) event when the drop-down portion is hidden. Use the [HideDropDownButton](#) property to hide the control's drop down button. Use the DropDown property to retrieve a boolean value that indicates whether the drop down portion of control is visible or hidden. Use the [AutoDropDown](#) property to specify whether the drop down portion of the control is shown when user starts type into it. If the Style property is DropDownList you can use the AutoDropDown property on True, and the list will be shown as soon as the user clicks the control's label area. Use the [HideDropDownButton](#) property to hide the drop down button. Use the [Background](#) property to change the drop down button's visual appearance.

The following VB sample displays the drop down portion of the control when it receives the focus:

```
Private Sub ComboBox1_GotFocus()  
    ComboBox1.DropDown() = True  
End Sub
```

The following VFP sample shows the drop down portion of the control when user clicks the form:

```
this.EXCOMB.Object.DropDown("") = .t.
```

where the "EXCOMB" is the name of the control in the VFP screen. In this case, if the Object property is missing it is possible that the VFP environment confuses the DropDown method with the [DropDown](#) event, and so an error could occur.

The following samples uses different methods to drop down the control when the user clicks the control's label area. In order to run any of the following sample, please insert the control to a form/dialog, and apply its default template (so we have loaded columns and items).

The following VB sample drops down the control when it receives the focus by clicking:

```
Private Declare Function GetMessagePos Lib "user32" () As Long
Private Declare Function WindowFromPoint Lib "user32" (ByVal xPoint As Long, ByVal
yPoint As Long) As Long
Private Declare Function IsChild Lib "user32" (ByVal hWndParent As Long, ByVal hwnd As
Long) As Long
```

```
Private Sub ComboBox1_GotFocus()
    With ComboBox1
        If (IsChild(.hwnd, WindowFromPoint(GetMessagePos() And &HFFF;, GetMessagePos()
/ 65536))) Then
            .DropDown("") = True
        End If
    End With
End Sub
```

The sample calls the DropDown method when the control gains the focus.

The following C++ sample shows the drop down portion of the control when the user clicks the control's label area (WM_PARENTNOTIFY method):

```
void OnParentNotify(UINT message, LPARAM lParam)
{
    CDialog::OnParentNotify(message, lParam);

    if ( LOWORD( message ) == WM_LBUTTONDOWN )
    {
        CRect rtClient;
        m_combobox.GetClientRect( &rtClient );
        m_combobox.ClientToScreen( &rtClient );
        ScreenToClient( &rtClient );
        POINT ptCursor = { ((long)(short)LOWORD(lParam)), ((long)
(short)HIWORD(lParam)) };
        if ( PtInRect( &rtClient, ptCursor ) )
        {
            COleVariant vtMissing; V_VT( &vtMissing )= VT_ERROR;
            m_combobox.SetDropDown( vtMissing, TRUE );
        }
    }
}
```

```
}
```

```
}
```

The OnParentNotify method handles the WM_PARENTNOTIFY message of the Dialog class. A parent's OnParentNotify member function is called by the framework when its child window is created or destroyed, or when the user clicks a mouse button while the cursor is over the child window. The m_combobox wraps the control, and it is a member of the dialog class. The sample calls the DropDown method only if the user clicks the control's label area.

The following C# sample displays the drop down portion of the control when user clicks the control's label area:

```
protected override void DefWndProc( ref Message msg )
{
    base.DefWndProc( ref msg );

    if ( msg.Msg == 0x210 /*WM_PARENTNOTIFY*/ )
    {
        Point pt = new Point( msg.LParam.ToInt32() & 0xFFFF, msg.LParam.ToInt32() >>
16 );
        PointToScreen( pt );
        Control c = GetChildAtPoint(pt);
        if ( c != null )
            if ( c == axComboBox1 )
                axComboBox1.set_DropDown( "", true );
    }
}
```

The following VB.NET sample displays the drop down portion of the control when the user clicks the control's label area:

```
Protected Overrides Sub DefWndProc(ByRef m As Message)
    MyBase.DefWndProc(m)

    If (m.Msg = &H210) Then 'WM_PARENTNOTIFY
        Dim pt As Point
        pt.X = m.LParam.ToInt32() And &HFFFF
```

```

pt.Y = m.LParam.ToInt32() / 65536
Dim c As Control
c = GetChildAtPoint(pt)
If Not (c Is Nothing) Then
    If c Is AxComboBox1 Then
        AxComboBox1.set_DropDown("", True)
    End If
End If
End If
End Sub

```

The following VFP sample displays the drop down message when the control gains the focus by clicking the control's label area (the code should be put on ComboBox1.GotFocus event):

```

DECLARE LONG GetMessagePos IN user32

DECLARE LONG WindowFromPoint IN user32;
    LONG x;;
    LONG y

DECLARE LONG IsChild IN user32;
    LONG hWndParent;;
    LONG hWnd

local x, y
x = Bitand(GetMessagePos(),4095)
y = Int(GetMessagePos() / 65536)
with thisform.ComboBox1
    if ( 0 # IsChild( .hWnd(), WindowFromPoint( x, y ) ) )
        .Object.DropDown("") = .t.
    endif
endwith

```

property ComboBox.DropDownBorder as AppearanceEnum

Specifies type of the border for the drop down portion.

Type

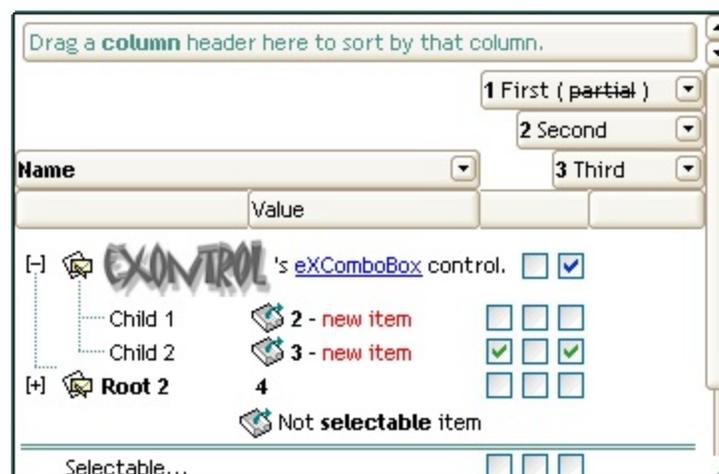
Description

[AppearanceEnum](#)

A AppearanceEnum expression that indicates the type of the border of the drop down portion of the control (where the list/tree is displayed), , 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 list's borders. For instance, if the DropDownBorder = 0x2000000, indicates that the second skin object in the Appearance collection defines the list's border. **The Client object in the skin, defines the client area of the drop down portion of the control. For instance, the list or tree, scrollbars are always displayed in the client area. The skin may contain transparent objects, and so you can define round corners for your drop down portion of the control. The [normal.ebn](#) file contains such of objects. Use the [eXButton](#)'s Skin builder to view or change this file.**

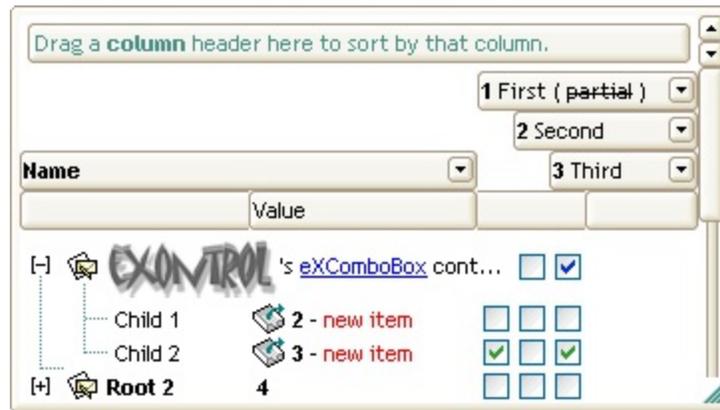
The DropDownBorder property defines the borders of the drop down portion of the control. Use the [Appearance](#) property to change the control's appearance. Use the [HeaderAppearance](#) property to change the control's header bar appearance. Use the [Style](#) property to specify the control's style. Use the [Add](#) method to add new skins to the control. Use the [BackColor](#) property to change the control's background color.

The following screen show shows the drop down portion of the control before running the sample:



The following screen show shows the drop down portion of the control when the [normal.ebn](#)

is applied to DropDownBorder property, as shown in the following samples:



The following VB sample changes the visual aspect of the borders of the drop down portion of the control:

```
With ComboBox1
    .BeginUpdate
        .VisualAppearance.Add &H16, "c:\temp\normal.ebn"
        .DropDownBorder = &H16000000
    .EndUpdate
End With
```

The following VB.NET sample changes the visual aspect of the borders of the drop down portion of the control:

```
With AxComboBox1
    .BeginUpdate()
    .VisualAppearance.Add(&H16, "c:\temp\normal.ebn")
    .DropDownBorder = &H16000000
    .EndUpdate()
End With
```

The following C# sample changes the visual aspect of the borders of the drop down portion of the control:

```
axComboBox1.BeginUpdate();
axComboBox1.VisualAppearance.Add(0x16, "c:\\temp\\normal.ebn");
axComboBox1.DropDownBorder = (EXCOMBOBOXLib.AppearanceEnum)0x16000000;
axComboBox1.EndUpdate();
```

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

```
m_comboBox.BeginUpdate();  
m_comboBox.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\normal.ebn" ) );  
m_comboBox.SetDropDownBorder( 0x16000000 );  
m_comboBox.EndUpdate();
```

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

```
with thisform.ComboBox1  
  .BeginUpdate  
    .VisualAppearance.Add(0x16, "c:\\temp\\normal.ebn")  
    .DropDownBorder = 0x16000000  
  .EndUpdate  
endwith
```

property ComboBox.DropDownButtonWidth as Long

Retrieves or sets the width, in pixels, to display the drop down button of the control (including the clear-button).

Type	Description
Long	A Long expression that specifies the width to display the drop down button.

By default, the DropDownButtonWidth property is -1, which indicates that the control's [ScrollWdth](#) property specifies the width of the drop down button. In other words, if the DropDownButtonWidth property is negative, the width of the drop down button and the vertical scroll bar are the same. If positive, it indicates the width, in pixels, to display the drop down button of the control. The property has effect only if the control's [Style](#) property is DropDown or DropDownList. Use the [HideDropDownButton](#) property to hide the control's drop down button when the control loses the focus.

property ComboBox.EditImage(ColIndex as Long) as Long

Specifies a value that indicates the index of icon being displayed on the column's edit box.

Type	Description
ColIndex as Long	A long expression that indicates the column's index.
Long	A long expression that indicates the index of the image in the Images collection being displayed on the column's edit box.

The EditImage property displays an icon in the column's edit box area. Use the [Images](#) method to assign a list of icons to the control. Use the [AssignEditImageOnSelect](#) property to automatically display the cell's icon in the column's edit box. The AssignEditImageOnSelect property has effect if the [Style](#) property is DropDown. By default, the EditImage property is 0. The Images collection is 1 based. Use the [CellImage](#) property to assign an icon to a cell. The control fires the [SelectionChanged](#) event when the user selects a new item.

property ComboBox.EditText([ColIndex as Variant]) as String

Retrieves or sets a value that indicates the edit's caption for a given 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. If -1, the EditText property accesses the text in the FilterFor field.
String	A string expression that indicates the text in the column's edit box.

The EditText property gets the text inside the column's edit box. If the [SingleEdit](#) property is True, then the [SearchColumnIndex](#) determines the index of the edit control that is displayed in the control. When the [AutoComplete](#) property is True, the text for each edit control is automatically updated with the values of the cells of the selected item. To find out the selected item you can call [SelectedItem](#) property of Items object. If the user has altered one of the control edit controls it fires the [EditChange](#) event. The EditText property is valid only if the control's [Style](#) property is not DropDownList. Use the [AssignEditImageOnSelect](#) property to display the cell's icon when user selects a new item. Use the [Value](#) property to get or set the selected value, when the control contains a single column. Use the [Select](#) property to get or set the selected value when control contains multiple columns. Use the [AutoSearch](#) property to disable incremental search feature in the control.

The following VB sample changes the control's s edit portion when the user clicks a button:

```
With ComboBox1
  Dim bSearch As Boolean
  bSearch = .AutoSearch
  .AutoSearch = False
  Dim bAutoComplete As Boolean
  bAutoComplete = .AutoComplete
  .AutoComplete = True
  .EditText = "test"
  .AutoSearch = bSearch
  .AutoComplete = bAutoComplete
End With
```

The following C++ sample changes the control's s edit portion when the user clicks a button:

```
BOOL bSearch = m_combobox.GetAutoSearch(), bAutoComplete =  
m_combobox.GetAutoComplete();  
m_combobox.SetAutoComplete( TRUE );  
m_combobox.SetAutoSearch( FALSE );  
m_combobox.SetEditText( COleVariant( long(0) ), "test" );  
m_combobox.SetAutoSearch( bSearch );  
m_combobox.SetAutoComplete( bAutoComplete );
```

The following VB.NET sample changes the control's s edit portion when the user clicks a button:

```
With AxComboBox1  
    Dim bSearch As Boolean = .AutoSearch  
    .AutoSearch = False  
    Dim bAutoComplete As Boolean = .AutoComplete  
    .AutoComplete = True  
    .set_EditText("test")  
    .AutoSearch = bSearch  
    .AutoComplete = bAutoComplete  
End With
```

The following C# sample changes the control's s edit portion when the user clicks a button:

```
Boolean bSearch = axComboBox1.AutoSearch;  
axComboBox1.AutoSearch = false;  
Boolean bAutoComplete = axComboBox1.AutoComplete;  
axComboBox1.AutoComplete = true;  
axComboBox1.set_EditText("test");  
axComboBox1.AutoSearch = bSearch;  
axComboBox1.AutoComplete = bAutoComplete;
```

The following VFP sample changes the control's s edit portion when the user clicks a button:

```
With thisform.ComboBox1  
    local bSearch, bAutoComplete  
    bSearch = .AutoSearch  
    .AutoSearch = .f.  
    bAutoComplete = .AutoComplete
```

```
.AutoComplete = .t.
```

```
.EditText(0) = "test"
```

```
.AutoSearch = bSearch
```

```
.AutoComplete = bAutoComplete
```

```
EndWith
```

property ComboBox.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 **ComboBox.EndUpdate ()**

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

Type

Description

The [BeginUpdate](#) method locks painting control, and the EndUpdate method unlocks painting the control. Once that BeginUpdate was invoked you should be sure that EndUpdate will be called when you done all operations into the combobox controls. The BeginUpdate and EndUpdate methods increases the speed of loading your items, by preventing painting the control when it suffers any change.

The following VB sample uses the BeginUpdate and EndUpdate methods to prevent painting during loading items 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

ComboBox1.BeginUpdate
For Each f In rs.Fields
    ComboBox1.Columns.Add f.Name
Next
ComboBox1.PutItems rs.GetRows()
ComboBox1.EndUpdate
```

The following VB sample prevents painting the control during loading columns and items:

```
With ComboBox1
    .BeginUpdate
    .Columns.Add ("Column 1")
    With .Items
        .AddItem "Item 1"
        .AddItem "Item 2"
    End With
    .EndUpdate
End With
```

The following C++ sample prevents painting the control during loading columns and items:

```
m_combobox.BeginUpdate();
CColumns columns = m_combobox.GetColumns();
columns.Add( "Column 1" );
CItems items = m_combobox.GetItems();
items.AddItem( COleVariant("Item 1") );
items.AddItem( COleVariant("Item 2") );
items.AddItem( COleVariant("Item 3") );
m_combobox.EndUpdate();
```

The following VB.NET sample prevents painting the control during loading columns and items:

```
With AxComboBox1
    .BeginUpdate()
    .Columns.Add("Column 1")
    With .Items
        .AddItem("Item 1")
        .AddItem("Item 2")
    End With
    .EndUpdate()
End With
```

The following C# sample prevents painting the control during loading columns and items:

```
axComboBox1.BeginUpdate();
axComboBox1.Columns.Add("Column 1");
axComboBox1.Items.AddItem("Item 1");
axComboBox1.Items.AddItem("Item 2");
axComboBox1.EndUpdate();
```

The following VFP sample prevents painting the control during loading columns and items:

```
With thisform.ComboBox1
    .BeginUpdate
    .Columns.Add ("Column 1")
    With .Items
        .AddItem("Item 1")
        .AddItem("Item 2")
    EndWith
```

.EndUpdate

EndWith

property ComboBox.EnsureOnSort as Boolean

Specifies whether the control ensures that the focused item fits the control's client area, when the user sorts the items.

Type	Description
Boolean	A boolean expression that indicates whether the control ensures that the focused item fits the control's client area after sorting the items.

By default, the EnsureOnSort property is True. If the EnsureOnSort property is True, the control calls the [EnsureVisibleItem](#) method to ensure that the focused item ([FocusItem](#) property) fits the control's client area, once items get sorted. Use the [SortOrder](#) property to sort a column. The [SortChildren](#) method sorts child items of an item. The EnsureOnSort property prevents scrolling of the control when child items are sorted.

property ComboBox.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 `ComboBox.ExecuteTemplate` (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed

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

Use the `ExecuteTemplate` property to returns the result of executing a template file. Use the [Template](#) property to execute a template without returning any result. Use the `ExecuteTemplate` property to execute code by passing instructions as a string (template string). For instance, you can use the `EXPRINT.PrintExt = CONTROL.ExecuteTemplate("me")` to print the control's content.

For instance, the following sample retrieves the the handle of the first visible item:

```
Debug.Print ComboBox1.ExecuteTemplate("Items.FirstVisibleItem()")
```

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

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

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

The Template script is composed by lines of instructions. Instructions are separated by

"\n\r" (newline) characters.

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- **variable = property**(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child")*
- **property**(list of arguments) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- **method**(list of arguments) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- **object.property**(list of arguments).**property**(list of arguments).... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property ComboBox.ExpandOnSearch as Boolean

Expands items automatically while user types characters to search for a specific item.

Type	Description
Boolean	A boolean expression that indicates whether the control expands items while user types characters to search for items.

Use the ExpandOnSearch property to expand items while user types characters to search for items using incremental search feature. Use the [AutoSearch](#) property to enable or disable incremental searching feature. Use the [AutoSearch](#) property of the [Column](#) object to specify the type of incremental searching being used within the column. The ExpandOnSearch property has no effect when the AutoSearch property is False. For instance, if the ExpandOnSearch property is True, the control fires the [BeforeExpandItem](#) event for items that have the [ItemHasChildren](#) property is True, when user types characters.

method `ComboBox.Export` ([Destination as Variant], [Options as Variant])

Exports the control's data to a CSV or HTML format.

Type	Description
Destination as Variant	<p>A String expression that specifies the file/format to be created. The Destination parameter indicates the format to be created as follows:</p> <ul style="list-style-type: none">• 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. <p>No error occurs, if the Export method can not create the file.</p>
Options as Variant	<p>A String expression that specifies the options to be used when exporting the control's data, as explained bellow.</p>
Return	Description
Variant	<p>A String expression that indicates the format being exported. It could be CSV or HTML format based on the Destination parameter.</p>

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 fourth 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 data 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:

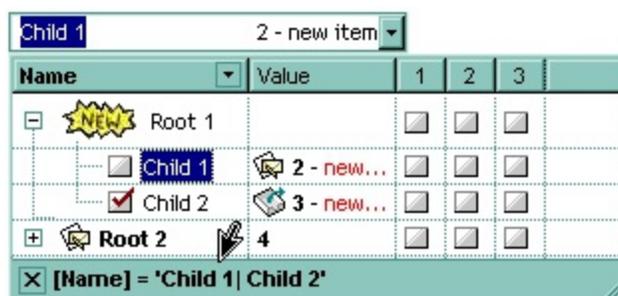
- HTML stands for HyperText Markup Language.
- is plain text using a character set such as ASCII, Unicode
- It's the way web pages are encoded to handle things like bold, italics and even color text red.

property ComboBox.FilterBarBackColor as Color

Specifies the background color of the control's filter bar. */*not supported in the lite version*/*

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 [FilterBarCaption](#) property to change the caption of the control's filter bar.



property ComboBox.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 below.

OrderID	ShipVia	Sel...	EmployeeID	OrderD...	Requir...	Shippe...	Freight	ShipName	ShipAddr...	ShipCity	ShipRegion	ShipPostal..
10251		<input type="checkbox"/>	4	10/1/2003	9/5/1994	8/15/1994	41.34	Victuailles...	2, rue du ...	Lyon		69004
10274		<input type="checkbox"/>	6	9/6/1994	10/4/1994	9/16/1994	6.01	Vins et alc...	59 rue de l...	Reims		51100
10260		<input type="checkbox"/>	4	8/19/1994	9/16/1994	8/29/1994	55.09	Ottilies Kä...	Mehrheim...	Köln		50739
10249		<input type="checkbox"/>	6	8/10/1994	9/16/1994	8/10/1994	11.61	Toms Spe...	Luisenstr. ...	Münster		44087
10284		<input type="checkbox"/>	4	9/19/1994	10/17/1994	9/27/1994	76.56	Lehmans...	Magazinw...	Frankfurt ...		60528
10267		<input type="checkbox"/>	4	8/29/1994	9/26/1994	9/6/1994	208.58	Frankenve...	Berliner Pl...	München		80805
10288		<input type="checkbox"/>	4	9/23/1994	10/21/1994	10/4/1994	7.45	Reggiani C...	Strada Pro...	Reggio Em...		42100
10281		<input type="checkbox"/>	4	9/14/1994	9/28/1994	9/21/1994	2.94	Romero y ...	Gran Vía, 1	Madrid		28001
10282		<input checked="" type="checkbox"/>	4	9/15/1994	10/13/1994	9/21/1994	12.69	Romero y ...	Gran Vía, 1	Madrid		28001
10269		<input type="checkbox"/>	5	8/31/1994	9/14/1994	9/9/1994	4.56	White Clov...	1029 - 12t...	Seattle	WA	98124
10296		<input type="checkbox"/>	6	10/4/1994	11/1/1994	10/12/1994	0.12	LILA-Supe...	Carrera 5...	Barquisim...	Lara	3508

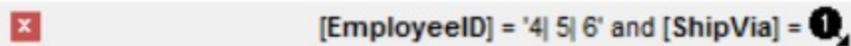
ShipCountry Select 11 result(s)"/>

For instance:

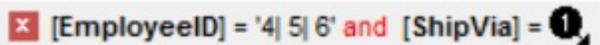
- "no filter", shows no filter caption all the time

- "" displays no filter bar, if no filter is applied, else it displays the current filter

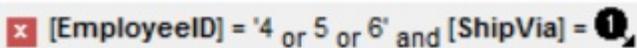
- "`<right>` + value", displays the current filter caption aligned to the right. You can include the exFilterBarShowCloseOnRight flag into the [FilterBarPromptVisible](#) property to display the close button aligned to the right



- "value replace ` and ` with `



- "value replace ` and ` with `



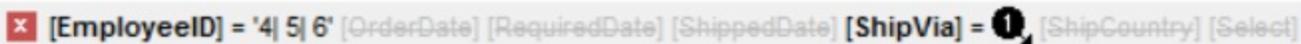
- "value replace `[` with `



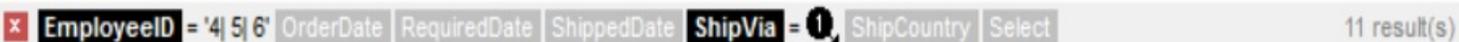
- "value + ` ` + available", displays the current filter, including all available columns to be filtered



- "allui" displays all available columns



- "((allui + `



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] = 1", so the control automatically applies HTML format, which you can change it. For instance, "upper(value)" displays the caption in uppercase or "value replace `` with `<fgcolor=808080>` replace `` with `</fgcolor>`" displays the column's name with a different foreground color.
- **itemcount** keyword returns the total number of items as indicated by [ItemCount](#) 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 ([ItemCount](#) property). If negative, a filter is applied, and the absolute value minus one, indicates the number of matching items after filter is applied. A matching item includes its parent items, if the control's [FilterInclude](#) property allows including child items. 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 matching 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>` + 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] = 1<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 `[` with `<bgcolor=000000><fgcolor=FFFFFF>` replace `` with `</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 `[` with `<bgcolor=000000><fgcolor=FFFFFF>` replace `` with `</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:

- ** bold ** bolds a part of the caption.
- **<u> underline </u>** specifies that the portion should appear as underlined.
- **<s> strikethrough </s>** specifies that the portion should appear as strikethrough.
- **<i> italic </i>** specifies that the portion should appear as italic.
- **<fgcolor=FF0000>fgcolor</fgcolor>** changes the foreground color for a portion.
- **<bgcolor=FF0000>bgcolor</bgcolor>** changes the background color for a portion.
- **
** breaks a line.
- **<solidline>** draws a solid line. It has no effect for a single line caption.
- **<dotline>** draws a dotted line. It has no effect for a single line caption.
- **<upline>** draws the line to the top of the text line
- **<r>** aligns the rest of the text line to the right side. It has no effect if the caption contains a single line.
- **number[:width]** inserts an icon inside the cell's caption. The number indicates the index of the icon being inserted. 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.
- **key[:width]** inserts a custom size picture being loaded using the [HTMLPicture](#) property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used
- **text ** 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.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **"** ("), **&#number**, For

instance, the `€` displays the EUR character, in UNICODE configuration. The `&` ampersand is only recognized as markup when it is followed by a known letter or a `#` character and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`;

Also, newer HTML format supports decorative text like follows:

- `<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 `rrggb`, `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 FFFFFFF;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 ComboBox.FilterBarDropDownHeight as Double

Specifies the height of the drop down filter window proportionally with the height of the control's list. */*not supported in the lite version*/*

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. The FilterBarDropDownHeight property is changed, if the user resizes at runt-time the drop down filter window. Use the [ShowFilter](#) method to show programmatically the column's 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 ComboBox1
    .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 ComboBox1
    .FilterBarDropDownHeight = 1
End With
```

The drop down filter window always include an item.

Child 1		2 - new item		
Name	Value	1	2	3
(NonBlanks)		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Root 1	2 - new ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Filter For:	3 - new ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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 ComboBox.FilterBarFont as IFontDisp

Retrieves or sets the font for control's filter bar. */*not supported in the lite version*/*

Type	Description
IFontDisp	A font object that indicates the font used to paint the description for control's filter

The FilterBarFont property defines the font in the filter bar area of the control. Use the [Font](#) property to change the control's font. Use the [ItemFont](#) property to assign a different font to an item. Use the [CellFont](#) property to assign a different font to the cell. You can change the control's font using options described [here](#). Use the , <s>, ... built-in HTML tags to display in bold, ... portions of text using the [HTMLCaption](#) property and [CellCaption](#) property as well. Use the [DisplayFilterButton](#) property to allow filter bar button in the column's header. If the [FilterFor](#) field is visible, the FilterBarFont property specifies the font being used on this field too.

The following VB sample changes the control's filter bar font:

```
With ComboBox1.FilterBarFont
    .Name = "Comic Sans MS"
    .Bold = True
End With
```

The following C++ sample changes the control's filter bar font:

```
#include "font.h"
COleFont font = m_combobox.GetFilterBarFont();
font.SetName("Comic Sans MS");
font.SetBold( TRUE );
```

The following VB.NET sample changes the control's filter bar font:

```
AxComboBox1.FilterBarFont = New System.Drawing.Font("Comic Sans MS", 10,
FontStyle.Bold)
```

The following C# sample changes the control's filter bar font:

```
axComboBox1.FilterBarFont = new System.Drawing.Font("Comic Sans MS", 10,
FontStyle.Bold);
```

The following VFP sample changes the control's filter bar font:

```
With thisform.ComboBox1.FilterBarFont
```

```
  .Name = "Comic Sans MS"
```

```
  .Bold = .t.
```

```
EndWith
```

property ComboBox.FilterBarForeColor as Color

Specifies the foreground color of the control's filter bar. */*not supported in the lite version*/*

Type	Description
Color	A color expression that defines the foreground color of the description of the control's filter.

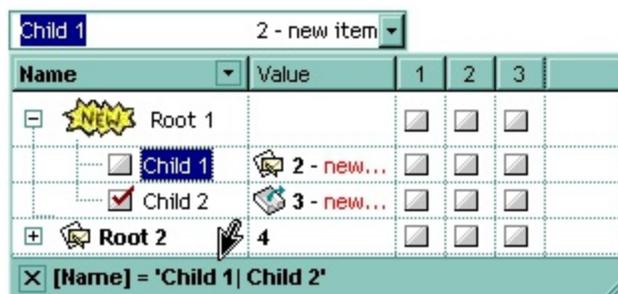
Use the FilterBarForeColor and [FilterBarBackColor](#) properties to define colors used to paint the description of the control's filter. Use the [FilterBarFont](#) property to specify the filter bar's font. Use the [FilterBarCaption](#) property to specify the caption of the control's filter bar.

property ComboBox.FilterBarHeight as Long

Specifies the height of the control's filter bar description. */*not supported in the lite version*/*

Type	Description
Long	A long expression that indicates the height of the filter bar status.

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 greater 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 change the caption of the control's filter bar. Use the [FilterBarFont](#) property to specify the font for the control's filter bar. Use the [FilterBarDropDownWidth](#) property to specify the width of the drop down filter window. The [FilterForVisible](#) property specifies whether the drop down portion of the control displays all the time a text-box editor to allow filtering items of the control. The [FilterForVisible](#) property specifies whether the drop down portion of the control displays all the time a text-box editor to allow filtering items of the control.



property ComboBox.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 "*<i><fgcolor=808080>Start Filter...</fgcolor></i>*". 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 [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**
- **<i> ... </i>** 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=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **number[:width]** 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.

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

The following screen show shows the filter prompt:

Name	Title	City
Nancy Davolio	Sales Representative	Seattle
Andrew Fuller	Vice President, Sales	Tacoma
Janet Leverling	Sales Representative	Kirkland
Margaret Peacock	Sales Representative	Redmond
Steven Buchanan	Sales Manager	London
Michael Suyama	Sales Representative	London
Robert King	Sales Representative	London
Laura Callahan	Inside Sales Coordinator	Seattle
Anne Dodsworth	Sales Representative	London

 Start Filter...

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 ComboBox.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 comboboxed 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 comboboxed 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 ComboBox.FilterForBackColor as Color

Retrieves or sets a value that indicates the FilterFor's background color.

Type	Description
Color	A Color expression that specifies the color to fill the FilterFor field. 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.

By default, the control's BackColor property indicates the FilterFor field. Use the [FilterForVisible](#) property to display a text box editor in the bottom side of the drop down portion of the control to allow filtering items in a different way. The [FilterForForeColor](#) property specifies the forecolor for the FilterFor field. The height of the FilterFor field is determined by the control's [FilterBarFont](#) property. The control fires the [EditChange\(-1\)](#) event when the user alter the text in the FilterFor field. The -1 parameter indicates the FilterFor field. Use the [EditText\(-1\)](#) property to specify the text being displayed in the FilterFor field.

property ComboBox.FilterForeColor as Color

Retrieves or sets a value that indicates the FilterFor's foreground color.

Type	Description
Color	A color expression that specifies the FilterFor field's foreground color.

By default, the control's ForeColor property indicates the FilterFor field. Use the [FilterForVisible](#) property to display a text box editor in the bottom side of the drop down portion of the control to allow filtering items in a different way. The [FilterForBackColor](#) property specifies the backcolor for the FilterFor field. The height of the FilterFor field is determined by the control's [FilterBarFont](#) property. The control fires the [EditChange\(-1\)](#) event when the user alter the text in the FilterFor field. The -1 parameter indicates the FilterFor field. Use the [EditText\(-1\)](#) property to specify the text being displayed in the FilterFor field.

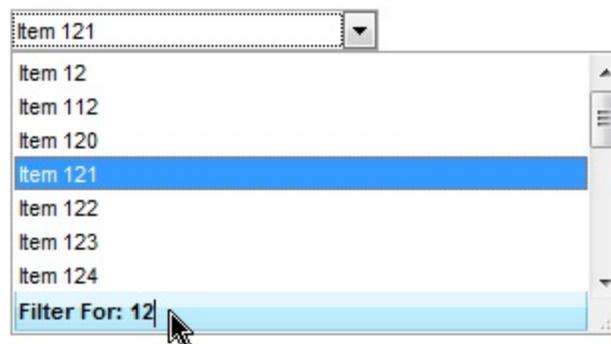
property ComboBox.FilterForVisible as Boolean

Specifies whether the control's FilterFor field is shown or hidden.

Type	Description
Boolean	A boolean expression that specifies whether the control shows or hides the FilterFor field.

By default, the FilterForVisible property is false, so the FilterFor field is hidden. Use the FilterForVisible property to display a text box editor in the bottom side of the drop down portion of the control to allow filtering items in a different way. The [Description\(exFilterForCaption\)](#) property specifies the prefix string (Filter For:) being shown in the FilterFor field. The [Description\(exFilterForTooltip\)](#) property specifies the tooltip to be shown when the cursor hovers the FilterFor field. The [FilterForBackColor](#) property specifies the FilterFor field background color or appearance, if using EBN technology. The [FilterForForeColor](#) property specifies the forecolor for the FilterFor field. The height of the FilterFor field is determined by the control's [FilterBarFont](#) property. The control fires the [EditChange\(-1\)](#) event when the user alter the text in the FilterFor field. The -1 parameter indicates the FilterFor field. Use the [EditText\(-1\)](#) property to specify the text being displayed in the FilterFor field. Use the [Filter](#), [FilterType](#) and [ApplyFilter](#) methods to filter the items in the drop down portion of the control as the user changes the text in the FilterFor field. Use the [FilterBarHeight](#) property on 0, to hide the description for the filter being applied. The CTRL + Backspace key removes the characters from start to the position of the cursor in the FilterFor field, while the CTRL + Delete key removes the characters from the position of the caret to the end of the field. The [DisplayFilterButton](#) property specifies whether the column's header displays a filter button, which allow filtering on multiple columns.

The following screen shows shows the FilterFor field:



The following VB sample changes the control's filter as soon as the user start typing characters in the FilterFor field:

```
Private Sub ComboBox1_EditChange(ByVal ColIndex As Long)
    If (ColIndex = -1) Then
        Dim sFilter As String
```

```
With ComboBox1
```

```
    sFilter = .EditText(-1)
```

```
    .BeginUpdate
```

```
    .ClearFilter
```

```
    .FilterBarHeight = 0
```

```
    If (Len(sFilter) > 0) Then
```

```
        With .Columns(0)
```

```
            .FilterType = exPattern
```

```
            .Filter = "*" + sFilter + "*"
```

```
        End With
```

```
    End If
```

```
    .ApplyFilter
```

```
    .EndUpdate
```

```
End With
```

```
End If
```

```
End Sub
```

property ComboBox.FilterInclude as FilterIncludeEnum

Specifies the items being included after the user applies the filter.

Type	Description
FilterIncludeEnum	A FilterIncludeEnum expression that indicates the items being included when the filter is applied.

By default, the FilterInclude property is `exItemsWithoutChilids`, which specifies that only items (and parent-items) that match the filter are being displayed. Use the FilterInclude property to specify whether the child-items should be displayed when the user applies the filter. Use the [Filter](#) property and [FilterType](#) property to specify the column's filter. Use the [ApplyFilter](#) to apply the filter at runtime. Use the [ClearFilter](#) method to clear the control's filter. Use the [FilterCriteria](#) property to filter items using the AND, OR and NOT operators. Use the [FilterBarPromptVisible](#) property to show the control's filter-prompt, that allows you to filter items as you type.

The following table shows items to display, when filter for "A" items, using different values for FilterInclude property:

no filter

`exItemsWithoutChilids`

`exItemsWithChilids`

`exRootsWithoutChilids`

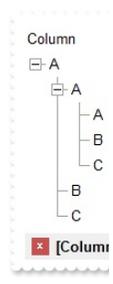
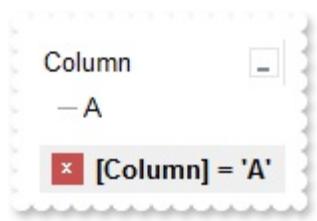
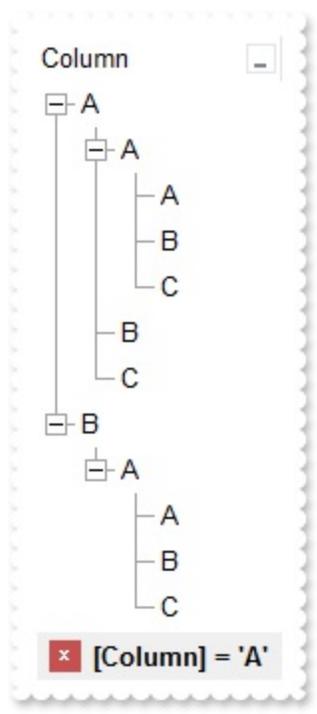
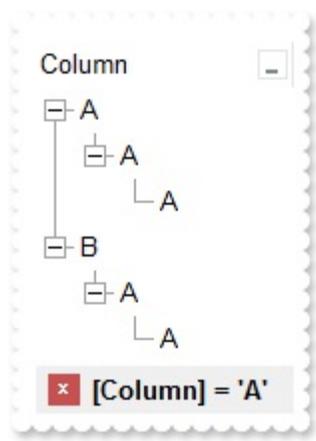
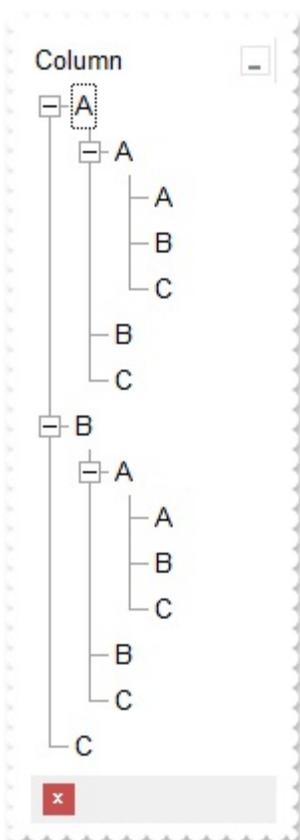
`exRootsW`

0

1

2

3



property ComboBox.FireClickOnSelect as Boolean

Fires Click event when the user changes the selection.

Type	Description
Boolean	A boolean expression that indicates whether the control fires the Click event when user changes the selection.

By default, the FireClickOnSelect property is False. The [Click](#) event is fired if the FireClickOnSelect property is True and the the user selects an item (using the keys or the mouse). The [SelectionChanged](#) is fired when the user selects a new item. Some containers (like old versions of the Microsoft Outlook) accepts only standard events like Click event. Use the [Value](#) property to get the selected value, in the column that pointed by the [SearchColumnIndex](#) property. Use the [Select](#) property to get or set the selected value on a specified column.

The following VB sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
Private Sub ComboBox1_Click()  
    Debug.Print ComboBox1.Value  
End Sub
```

The following C++ sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
void OnClickCombobox1()  
{  
    OutputDebugString( V2S( &m_combobox.GetValue() ) );  
}
```

where the V2S function converts a VARIANT value 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 the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
Private Sub AxComboBox1_ClickEvent(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxComboBox1.ClickEvent  
    Debug.WriteLine(AxComboBox1.Value)  
End Sub
```

The following C# sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
private void axComboBox1_ClickEvent(object sender, EventArgs e)  
{  
    System.Diagnostics.Debug.WriteLine(axComboBox1.Value.ToString());  
}
```

The following VFP sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
*** ActiveX Control Event ***  
  
wait window nowait thisform.ComboBox1.Object.Value
```

property ComboBox.FireNotInList as Boolean

Specifies whether the control fires the NotInList event.

Type	Description
Boolean	A boolean expression that specifies whether the control fires NotInList event.

By default, the FireNotInList event is False. The [NotInList](#) event notifies your application that the user enters a value in the text box portion of the combo box (label) that isn't in the combo box list. The NotInList event may **not** be fired if the [AutoComplete](#) property is True.

Before running any of the following samples please make sure that the Style property is DropDown, FireNotInList property is True, AutoComplete property is False.

The following VB sample adds a new item, if it is not in the combo box list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
Private Sub ComboBox1_NotInList()  
    With ComboBox1  
        Dim h As EXCOMBOBOXLibCtl.HITEM  
        h = .Items.AddItem(.EditText(0))  
        .Items.SelectItem(h) = True  
    End With  
End Sub
```

```
Private Sub Form_Load()  
    With ComboBox1  
        .BeginUpdate  
        .AutoComplete = False  
        .FireNotInList = True  
        .ColumnAutoResize = True  
        .HeaderVisible = False  
        .Columns.Add "Column"  
        With .Items  
            .AddItem "Ana"  
            .AddItem "Maria"  
        End With  
        .EndUpdate  
    End With
```

End Sub

The following JScript sample adds a new item if the control's drop down list doesn't contain it :

```
<SCRIPT FOR="ComboBox1" EVENT="NotInList()" LANGUAGE="javascript">  
  
obj = document.getElementById ( "ComboBox1" );  
h = obj.Items.AddItem(obj.EditText(0));  
obj.Items.SelectItem(h) = true  
  
</SCRIPT>
```

The following C++ sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
void OnNotInListCombobox1()  
{  
    CItems items = m_combobox.GetItems();  
    long hItem = items.AddItem( COleVariant(  
m_combobox.GetEditText(COleVariant(long(0))) ) );  
    items.SetSelectItem( hItem, TRUE );  
}
```

The following VB.NET sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
Private Sub AxComboBox1_NotInList(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxComboBox1.NotInList  
    With AxComboBox1  
        Dim h As Integer = .Items.AddItem(.get_EditText(0))  
        .Items.SelectItem(h) = True  
    End With  
End Sub
```

The following C# sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
private void axComboBox1_NotInList(object sender, EventArgs e)  
{
```

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;  
int hItem = items.AddItem(axComboBox1.get_EditText(0));  
items.set_SelectItem(hItem, true);  
}
```

The following VFP sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

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

```
With thisform.ComboBox1  
  .Items.DefaultItem = .Items.AddItem(.EditText(0))  
  .Items.SelectItem(0) = .t.  
EndWith
```

property ComboBox.Font as IFontDisp

Retrieves or sets the control's font.

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

Use the Font property to change the control's font. Use the [ItemFont](#) property to assign a different font to an item. Use the [CellFont](#) property to assign a different font to the cell. You can change the control's font using options described [here](#). Use the , <s>, ... built-in HTML tags to display in bold, ... portions of text using the [HTMLCaption](#) property and [CellCaption](#) property as well. Use the [FilterBarFont](#) property to change the control's filter bar font. Use the Font property to assign a font to the control at design time. The [Locked](#) property recreates the edit controls in the control's label area, if the [Style](#) property is DropDown.

The following VB sample changes the control's font at runtime:

```
With ComboBox1.Font
    .Name = "Comic Sans MS"
    .Bold = True
End With
ComboBox1.Locked = ComboBox1.Locked
```

The following C++ sample changes the control's font at runtime:

```
#include "font.h"
COleFont font = m_combobox.GetFont();
font.SetName("Comic Sans MS");
font.SetBold( TRUE );
m_combobox.SetLocked( m_combobox.GetLocked() );
```

The following VB.NET sample changes the control's font at runtime:

```
AxComboBox1.Font = New System.Drawing.Font("Comic Sans MS", 10, FontStyle.Bold)
AxComboBox1.Locked = AxComboBox1.Locked
```

The following C# sample changes the control's font at runtime:

```
axComboBox1.Font = new System.Drawing.Font("Comic Sans MS", 10, FontStyle.Bold);
axComboBox1.Locked = axComboBox1.Locked;
```

The following VFP sample changes the control's font at runtime:

```
With thisform.ComboBox1.Font
```

```
  .Name = "Comic Sans MS"
```

```
  .Bold = .t.
```

```
EndWith
```

```
thisform.ComboBox1.Locked = thisform.ComboBox1.Locked
```

property ComboBox.ForeColor as Color

Retrieves or sets a value that indicates the control's foreground color.

Type	Description
Color	A color expression that indicates the color used by control to paint the scrolled area.

The ForeColor property changes the foreground color of the control's scrolled area. The ExComboBox 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. To change the background color of the control's locked area use [BackColorLock](#) property. Use the [ForeColorEdit](#) property to specify the foreground color of the control's edit box. Use the [CellForeColor](#) property to specify the cell's foreground color. Use the [ItemForeColor](#) property to specify the item's foreground color. Use the [HeaderBackColor](#) and [HeaderForeColor](#) properties to defines the colors for the control's header bar.

property ComboBox.ForeColorEdit as Color

Specifies the control's edit foreground color.

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

Use the ForeColorEdit property to specify the foreground color of the control's edit box. Use the [ForeColor](#) property to specify the foreground color of the control's drop down portion. Use the [CellForeColor](#) property to specify the cell's foreground color. Use the [ItemForeColor](#) property to specify the item's foreground color. The BackColorEdit property has effect only, if the [Style](#) property is not DropDownList.

property ComboBox.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 locked area.

The ExComboBox 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 area contains the columns that can be scrolled horizontally. To change the background color of the control's locked area use [BackColorLock](#) property. Use the [BackColorLock](#) property to specify the background color for locked columns.

property ComboBox.ForeColorSortBar as Color

Retrieves or sets a value that indicates the sort bar's foreground color. */*not supported in the lite version*/*

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 [HeaderForeColor](#) property to specify the background color of the control's header bar.

method ComboBox.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 ComboBox.

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 ComboBox.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. Use the [AnchorFromPoint](#) property to retrieve the identifier of the anchor element from the cursor.

method `ComboBox.FreezeEvents` (Freeze as Boolean)

Prevents the control to fire any event.

Type	Description
Freeze as Boolean	A Boolean expression that specifies whether the control's events are froze or unfroze

The `FreezeEvents(True)` method freezes the control's events until the `FreezeEvents(False)` method is called. You can use the `FreezeEvents` method to improve performance of the control while loading data into it.

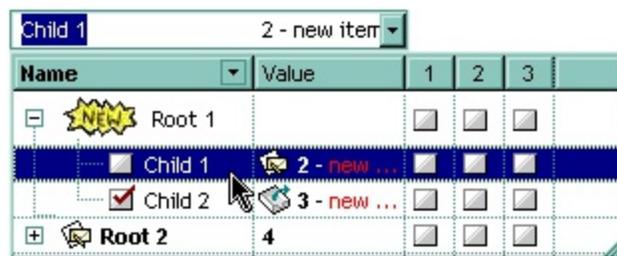
property ComboBox.FullRowSelect as Boolean

Enables full-row selection in the control.

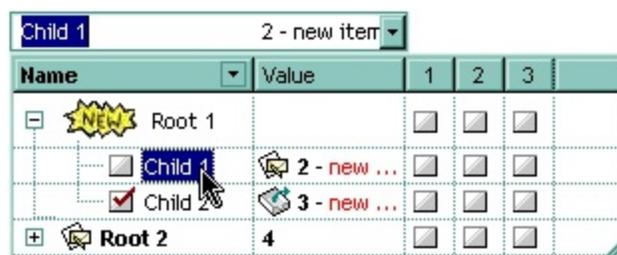
Type	Description
Boolean	A boolean expression that indicates whether the control supports full-row selection.

The FullRowSelect property specifies whether the selection spans the entire width of the control. If the FullRowSelect property is True, the entire line is highlighted. If the FullRowSelect property is False, only the cell pointed to by the [TreeColumnIndex](#) property is highlighted in the selected item. Use the [SelectedItem](#) property to select programmatically an item. Use the [SelectedItem](#) property to retrieve the selected item. Use the [Select](#) property to select an item given a value on a specified column. Use the [Value](#) property to select an item given its value on a single column control. Use the [SelForeColor](#) and [SelBackColor](#) properties to specify the colors to highlight the selected item.

The following screen shot shows the control when the FullRowSelect property is True.



The following screen shot shows the control when the FullRowSelect property is False.



method ComboBox.GetItems (Options as Variant)

Gets the collection of items into a safe array,

Type	Description
Options as Variant	<p>Specifies a long expression as follows:</p> <ul style="list-style-type: none">• if 0, the result is a two-dimensional array with cell's captions. The list includes the <i>collapsed</i> items, and the items are included as they are displayed (sorted, filtered). This option exports the captions of cells. This option exports the captions of the cells (CellCaption property)• if 1, the result the one-dimensional array of handles of items in the control as they are displayed (sorted, filtered). The list <i>does not include the collapsed items</i>. For instance, the first element in the array indicates the handle of the first item in the control, which can be different that FirstVisibleItem result, even if the control is vertically scrolled. This option exports the handles of the items. For instance, you can use the ItemToIndex property to get the index of the item based on its handle.• else if other, and the number of columns is 1, the result is a one-dimensional array that includes the items and its child items as they are displayed (sorted, filtered). In this case, the array may contains other arrays that specifies the child items. The list includes the <i>collapsed</i> items, and the items are included as they are displayed (sorted, filtered). This option exports the captions of the cells (CellCaption property)

If missing, the Options parameter is 0. If the control displays no items, the result is an empty object (VT_EMPTY).

Return	Description
Variant	A safe array that holds the items in the control. If the control has a single column, the GetItems returns a single dimension array (object[]), else The safe array being returned has two dimensions (object[,]). 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. Also, the `GetItems` method collect the child items as well, no matter if the parent item is collapsed. 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 handles 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.

/NET Assembly:

The following C# sample converts the returned value to a `object[]` when the control contains a single column:

```
object[] Items = (object[])excombobox1.GetItems()
```

or when the control contains multiple columns, the syntax is as follows:

```
object[,] Items = (object[,])excombobox1.GetItems()
```

The following VB.NET sample converts the returned value to a `Object()` when the control contains a single column:

```
Dim Items As Object() = Excombobox1.GetItems()
```

or when the control contains multiple columns, the syntax is as follows:

```
Dim Items As Object(,) = Excombobox1.GetItems()
```

/COM version:

The following VB sample gets the items from a control and put them to the second one:

```
With ComboBox2
    .BeginUpdate
    .Columns.Clear
    Dim c As EXCOMBOBOXLibCtl.Column
    For Each c In ComboBox1.Columns
        .Columns.Add c.Caption
    Next
    .PutItems ComboBox1.GetItems
```

.EndUpdate

End With

The following C++ sample gets the items from a control and put them to the second one:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_combobox2.BeginUpdate();
    CColumns columns = m_combobox.GetColumns(), columns2 =
m_combobox2.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_combobox.GetItems( vtMissing );
m_combobox2.PutItems( &vtItems, vtMissing );
m_combobox2.EndUpdate();
```

The following C# sample gets the items from a control and put them to a second one:

```
axComboBox2.BeginUpdate();
for (int i = 0; i < axComboBox1.Columns.Count; i++)
    axComboBox2.Columns.Add(axComboBox1.Columns[i].Caption);
object vtItems = axComboBox1.GetItems("");
axComboBox2.PutItems(ref vtItems);
axComboBox2.EndUpdate();
```

The following VB.NET sample gets the items from a control and put them to a second one:

```
With AxComboBox2
    .BeginUpdate()
    Dim j As Integer
    For j = 0 To AxComboBox1.Columns.Count - 1
        .Columns.Add(AxComboBox1.Columns(j).Caption)
    Next
    Dim vtItems As Object
    vtItems = AxComboBox1.GetItems("")
    .PutItems(vtItems)
    .EndUpdate()
```

The following VFP sample gets the items from a control and put them to a second one:

```
local i
with thisform.ComboBox2
  .BeginUpdate()
  for i = 0 to thisform.ComboBox1.Columns.Count - 1
    .Columns.Add( thisform.ComboBox1.Columns(i).Caption )
  next
  local array vtItems[1]
  vtItems = thisform.ComboBox1.GetItems("")
  .PutItems( @vtItems )
  .EndUpdate()
endwith
```

property ComboBox.GridLineStyle as GridLineStyleEnum

Specifies the style for gridlines in the list part of the control.

Type	Description
GridLineStyleEnum	A GridLineStyleEnum 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. Use the [LinesAtRoot](#) property specifies whether the control links the root items of the control. Use the [HasLines](#) property to specify whether the control draws the link between child items to their corresponding parent item.

The following VB sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = GridLineStyleEnum.exGridLinesHDash Or  
GridLineStyleEnum.exGridLinesVSolid
```

The following VB/NET sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXCOMBOBOXLib.GridLineStyleEnum.exGridLinesHDash Or  
exontrol.EXCOMBOBOXLib.GridLineStyleEnum.exGridLinesVSolid
```

The following C# sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXCOMBOBOXLib.GridLineStyleEnum.exGridLinesHDash |  
exontrol.EXCOMBOBOXLib.GridLineStyleEnum.exGridLinesVSolid;
```

The following Delphi sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle := Integer(EXCOMBOBOXLib.GridLineStyleEnum.exGridLinesHDash) Or  
Integer(EXCOMBOBOXLib.GridLineStyleEnum.exGridLinesVSolid);
```

The following VFP sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = 36
```


property ComboBox.HasButtons as ExpandButtonEnum

Adds a button to the left side of each parent item.

Type	Description
ExpandButtonEnum	An ExpandButtonEnum expression that indicates whether the left side button of each parent item is visible or hidden.

The HasButtons property has effect only if the data is displayed as a tree. Use the [InsertItem](#) method to insert child items. The control displays a +/- button to parent items, if the HasButtons property is not zero, the [ItemChild](#) property is not empty, or the [ItemHasChildren](#) property is True. The user can click the +/- button to expand or collapse the child items. Use the [ExpandItem](#) property of [Items](#) object to programmatically expand/collapse an item. The [HasButtonsCustom](#) property specifies the index of icons being used for +/- signs on parent items, when HasButtons property is exCustom.

The following VB sample changes the +/- button appearance:

```
With ComboBox1
    .HasButtons = ExpandButtonEnum.exWPlus
End With
```

The following C++ sample changes the +/- button appearance:

```
m_combobox.SetHasButtons( 3 /*exWPlus*/ );
```

The following VB.NET sample changes the +/- button appearance:

```
With AxComboBox1
    .HasButtons = EXCOMBOBOXLib.ExpandButtonEnum.exWPlus
End With
```

The following C# sample changes the +/- button appearance:

```
axComboBox1.HasButtons = EXCOMBOBOXLib.ExpandButtonEnum.exWPlus;
```

The following VFP sample changes the +/- button appearance:

```
with thisform.ComboBox1
    .HasButtons = 3 && exWPlus
endwith
```


property ComboBox.HasButtonsCustom(Expanded as Boolean) as Long

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

Type	Description
Expanded as Boolean	A boolean expression that indicates the sign being changed.
Long	A long expression that indicates the icon being used for +/- signs on the parent items. 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 HasButtonsCustom property to assign custom icons to the +/- signs on the parent items. The HasButtonsCustom property has effect only if the [HasButtons](#) property is exCustom. Use the [Images](#), [Replacelcon](#) methods to add new icons to the control.

property ComboBox.HasLines as HierarchyLineEnum

Enhances the graphic representation of a tree control's hierarchy by drawing lines that link child items to their corresponding parent item.

Type	Description
HierarchyLineEnum	An HierarchyLinesEnum expression that indicates whether the control displays the hierarchy lines.

Use the HasLines property to hide the hierarchy lines. Use the [LinesAtRoot](#) property to allow control displays a line that links that root items of the control. Use the [InsertItem](#) method to insert new items to the control. Use [HasButtons](#) property to hide the buttons displayed at the left of each parent item. Use the [DrawGridLines](#) property to display grid lines.

property ComboBox.HeaderAppearance as AppearanceEnum

Retrieves or sets a value that indicates the header's appearance.

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's header bar appearance.

Use [HeaderVisible](#) property to hide 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 [BackColorHeader](#) property to change the header's background color, or the visual appearance for the columns in the control's header.

property ComboBox.HeaderBackColor 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 HeaderBackColor and [HeaderForeColor](#) properties to defines the colors for the control's header bar. Use the [Def\(exHeaderBackColor\)](#) property to change the background color or the visual appearance for a particular column, in the header area. If the [Def\(exHeaderForeColor\)](#) property is not zero, it defines the foreground color to paint the column's caption in the header area. Use the [LevelKey](#) property to display the control's header bar using multiple levels. Use the [HeaderVisible](#) property to specify whether the control's header bar is visible or hidden. 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 [BackColor](#) property to specify the control's background color.

property ComboBox.HeaderForeColor as Color

Specifies the header's foreground color.

Type	Description
Color	A color expression that defines the foreground color for control's header bar.

Use the HeaderForeColor and [HeaderBackColor](#) properties to defines the colors of the control's header bar. If the [Def\(exHeaderForeColor\)](#) property is not zero, it defines the foreground color to paint the column's caption in the header area. 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. Use the [HeaderVisible](#) property to specify whether the control's header bar is visible or hidden. The control's header displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the [ForeColor](#) property to specify the control's foreground color.

property ComboBox.HeaderHeight as Long

Retrieves or sets a value indicating control's header height.

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

The HeaderHeight property specifies the height of the columns header bar in pixels. Use the [HeaderVisible](#) property to hide the control's header bar. Use the HeaderHeight property to specify the height of 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 [HeaderBackColor](#) and [HeaderForeColor](#) properties to defines the colors for the control's header bar. *If the [HeaderSingleLine](#) property is False, the HeaderHeight property specifies the maximum height of the control's header when the user resizes the columns.*

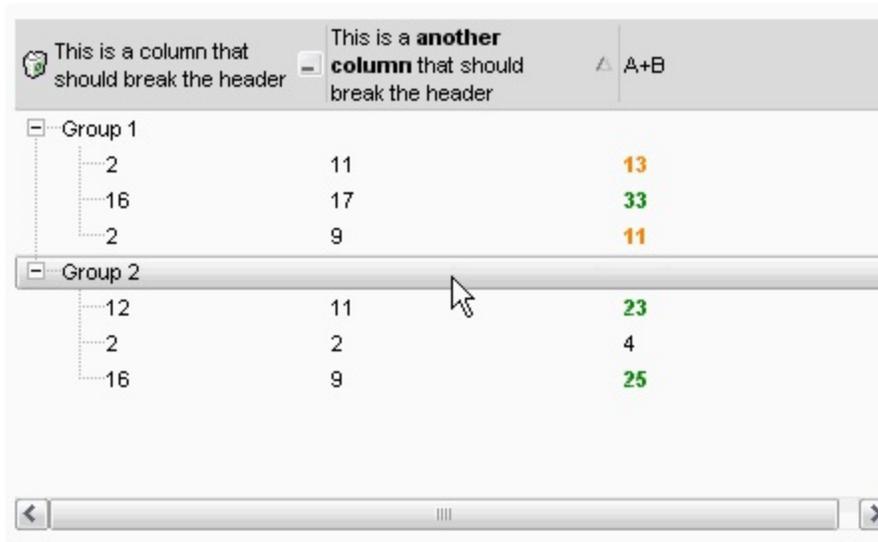
property ComboBox.HeaderSingleLine as Boolean

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 default, 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.

The following screen show shows the control's header while it displays a multiple lines (HeaderSingleLine = False):



The screenshot shows a data grid control with a header and two data groups. The header has three columns. The first column has a caption that wraps onto two lines: "This is a column that should break the header". The second column has a caption that also wraps onto two lines: "This is a **another** column that should break the header". The third column has a caption "A+B". The data is organized into two groups. Group 1 has three rows with values (2, 11, 13), (16, 17, 33), and (2, 9, 11). Group 2 has three rows with values (12, 11, 23), (2, 2, 4), and (16, 9, 25). A mouse cursor is pointing at the second column of the first row in Group 2.

This is a column that should break the header	This is a another column that should break the header	A+B
Group 1		
2	11	13
16	17	33
2	9	11
Group 2		
12	11	23
2	2	4
16	9	25

The following screen shot shows the control's header on multiple levels using the [LevelKey](#) property:

Level 1		
Level 2		
This is a colu...	This is a another colu...	A+B
Level 3		
Group 1		
2	11	13
16	17	33
2	9	11
Group 2		
12	11	23
2	2	4
16	9	25

The following screen show shows the control's header while it displays a single line (`HeaderSingleLine = True`):

This is a column that s...			This is a another column ..	A+B
Group 1				
2	11		13	
16	17		33	
2	9		11	
Group 2				
12	11	↔	23	
2	2		4	
16	9		25	

property ComboBox.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 control's header bar is visible. Use the HeaderVisible property to hide the control's header bar. Use the [HeaderAppearance](#) to change the control's header bar appearance. Use the [HeaderHeight](#) property to specify the height of the control's header bar. Use the [BackColorLevelHeader](#) property to specify the background color of the header when it displays multiple levels. Use the [HeaderBackColor](#) and [HeaderForeColor](#) properties to defines the colors for the control's header bar. Use the [Add](#) method to add new columns to the control.

property ComboBox.HeightList([Reserved as Variant]) as Long

Specifies the height of the control's drop-down window.

Type	Description
Reserved as Variant	Reserved.
Long	A long expression that specifies the height of the control's drop-down window.

The HeightList property specifies the height of the control's drop down window. The HeightList property has no effect if the [Style](#) property is Simple. Use the [WidthList](#) property to specify the width of the control's drop down window. Use the [MinHeightList](#) property to specify the minimum height of the control's drop down window. Use the [IntegralHeight](#) property to avoid showing partial items. Use the [LabelHeight](#) property to specify the height of the control's label area. Use the [AllowSizeGrip](#) property to specify whether the drop down portion of the control is resizable at runtime.

The following VB sample specifies the drop down height:

```
ComboBox1.HeightList() = 100
```

The following C++ sample specifies the height of the drop down portion of the control:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;  
m_combobox.SetHeightList( vtMissing, 100 );
```

The following VB.NET sample specifies the height of the drop down portion of the control:

```
AxComboBox1.set_HeightList(100)
```

The following C# sample specifies the height of the drop down portion of the control:

```
axComboBox1.set_HeightList(100);
```

The following VFP sample specifies the height of the drop down portion of the control:

```
with thisform.ComboBox1.Object  
    .HeightList("") = 100  
endwith
```

property ComboBox.HideDropDownButton as Long

Returns a value that determines whether the drop down button is hidden when the control loses the focus.

Type	Description
Long	A long expression that indicates whether the control hides the drop down button. The valid values are: 0 (default) - the control shows always the drop down button, -1 - the control hides the drop down button when control loses the focus. 1 - the control hides the drop down button.

By default, the HideDropDownButton property is 0. Use the HideDropDownButton property to hide the control's drop down button when the control loses the focus. The property has effect only if the control's [Style](#) property is DropDown or DropDownList. The HideDropDownButton property has no effect if the control's Style property is Simple. Use the [UseTabKey](#) property to control who handles the TAB key, the control or the form/container that hosts the control. Use the [DropDown](#) method to programmatically drop down the control. Use the [Background](#) property to change the drop down button's visual appearance. Use the [DropDownButtonWidth](#) property to specify the width in pixels of the drop down button.

If the HideDropDownButton property is:

- 0 (default), the control shows always the drop down button
- 1, the control do not show the drop down button
- -1, the control shows the drop down button when the control gain the focus, and hides the drop down button when the control loses the focus

property ComboBox.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none">• 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), <p>If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added</p>

The HTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the 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 "pic1" 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 ComboBox.hWnd as Long

Retrieves the control's window handle.

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

Use the hWnd property to get the control's main window handle. Use the [hWndDropDown](#) property to retrieve the handle of the drop down window. The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

property ComboBox.hWndDropDown as Long

Retrieves a value that indicates the drop down window's handle.

Type	Description
Long	A long expression that indicates the drop down window's handle.

The hWndDropDown property is invalid while the control's [Style](#) property is 0 (Simple Combo). Use the [hWnd](#) property to retrieves the handle of the control's main window. The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

method ComboBox.Images (Handle as Variant)

Sets at runtime the control's image list.

Type

Description

The Handle parameter can be:

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

Handle as Variant

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

The control provides an image list window, that's displayed at design time. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. Use the [ShowImageList](#) property to hide the image list window, at design time. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list window. At runtime, the user can use the Images and [Replacelcon](#) method to change the Images collection. The Images collection is 1 based. Use the [CellImage](#) property to assign an icon to a cell. Use the [CellImages](#) property to assign multiple icons to a cell. Use the [HeaderImage](#) property to assign an icon to the column's header.

The following VB sample replaces the entire list of icons (if it was loaded at design time), using a Microsoft Image List control (ImageList1):

```
ComboBox1.Images ImageList1.hImageList
```

The following VB sample initializes the Images collection using BASE64 encoded string:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellImage(Item, 0) = 1 + .ItemToIndex(Item) Mod 3
    End With
End Sub
```

```
Private Sub Form_Load()
    With ComboBox1
        .BeginUpdate
        .Images
"gBJJgBAIDAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

        .ColumnAutoResize = True
        .HeaderVisible = False
        .Columns.Add "Column 1"

        .PutItems Array("Item 1", "Item 2", "Item 3")
    End With
End Sub
```

.EndUpdate

End With

End Sub

The following C++ sample initializes the Images collection using BASE64 encoded string:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_combobox.BeginUpdate();
CString s =
"gBJJgBAIDAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn
s +=
"/xoAw9ZiFdxBAAGVxM5yOTzkPy+MzGRpmdx2kl2epGY1WgxmZl+Yyery2yyGHyeirGoo+C
s +=
"NbDDLO2xz5PIBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwyu0UPS/U
s +=
"kSSAAkqUU2nE20gmp5oo6JwH+eZ31EjJwB+eBn1K/AHnBWIfvwAZwACYAHsMy9clMyFeF
m_combobox.Images( COleVariant( s ) );
m_combobox.GetColumns().Add( "Column 1" );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItems items = m_combobox.GetItems();
long h = items.AddItem( COleVariant( "Item 1" ) );
items.SetCellImage( COleVariant( h ), COleVariant( (long) 0 ), 1 );
h = items.AddItem( COleVariant( "Item 2" ) );
items.SetCellImages( COleVariant( h ), COleVariant( (long) 0 ), COleVariant( "2,3" ) );
m_combobox.EndUpdate();
```

The following VB.NET sample initializes the Images collection using BASE64 encoded string:

```
Dim s As String
With AxComboBox1
    .BeginUpdate()
```

```

s =
"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrkltl0vmExn

s = s + "Poyf5xoojKAg"
.Images(s)

.Columns.Add("Column 1")
With .Items
    Dim h As Integer
    h = .AddItem("Item 1")
    .CellImage(h, 0) = 1
    h = .AddItem("Item 2")
    .CellImages(h, 0) = "2,3"
End With
.EndUpdate()
End With

```

The following C# sample initializes the Images collection using BASE64 encoded string:

```

axComboBox1.BeginUpdate();
string s =
"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrkltl0vmExn

s = s + "Poyf5xoojKAg";
axComboBox1.Images(s);
axComboBox1.Columns.Add("Column 1");
int h = axComboBox1.Items.AddItem("Item 1");
axComboBox1.Items.set_CellImage(h, 0, 1);
h = axComboBox1.Items.AddItem("Item 2");
axComboBox1.Items.set_CellImages(h, 0, "2,3");
axComboBox1.EndUpdate();

```

The following VFP sample initializes the Images collection using BASE64 encoded string:

```

local s
With thisform.ComboBox1
    .BeginUpdate()
    s =

```

"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFElEaEEaAlAkcbk0oIUrlktl0vmExn

s = s +

"dr1fsFhsVjslls1ntFptVrtltt1vuFxuVzul1u13vF5vV7vl9v1/wGBnqAQEZwmCxFhYGLib/xoAw9

s = s +

"Goo+03mM02Jzee029y2Ewum2+FnOTlGezHNx0b3/C3U258a4mP5HVvOw52s2fg2vH6ml

s = s +

"kicAJnCbsNbDDLO2xz5PIBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rw

s = s +

"wi/8iNPJMjSo0clvjMLuTHLkJTNCqVTSms2Tkq8jTzOcvP/BsePUocQLDQ9AJ3LtfUbr1Hqaiw

s = s + "Poyf5xoojKAg"

.Images(s)

.Columns.Add("Column 1")

With .Items

.DefaultItem = .AddItem("Item 1")

.CellImage(0, 0) = 1

.DefaultItem = .AddItem("Item 2")

.CellImages(0, 0) = "2,3"

EndWith

.EndUpdate()

EndWith

property ComboBox.ImageSize as Long

Retrieves or sets the size of icons the control displays.

Type	Description
Long	A long expression that defines the size of icons the control displays.

By default, the ImageSize property is 16 (pixels). The ImageSize property specifies the size of icons being loaded using the [Images](#) method. The control's Images collection is cleared if the ImageSize property is changed, so it is recommended to set the ImageSize property before calling the Images method. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. For instance, if the ICO file to load includes different types the one closest with the size specified by ImageSize property is loaded by Images method. The ImageSize property does NOT change the height for the control's font.

The ImageSize property defines the size to display the following UI elements:

- any icon that a cell or column displays (`number` ex-html tag, [CellImage](#), [CellImages](#))
- check-box or radio-buttons ([CellHasCheckBox](#), [CellHasRadioButton](#))
- expand/collapse glyphs ([HasButtons](#), [HasButtonsCustom](#))
- header's sorting or drop down-filter glyphs

property ComboBox.Indent as Long

Retrieves or sets the amount, in pixels, that child items are indented relative to their parent items.

Type	Description
Long	A long expression that indicates the amount, in pixels, that child items are indented relative to their parent items.

If the Indent property is 0, the child items are not indented relative to their parent item. Use [HasLines](#) and [LinesAtRoot](#) properties to show the hierarchy lines. Use the [HasButtons](#) property to define the +/- signs appearance. Use the [TreeColumnIndex](#) property to define the index of the column that displays the hierarchy. Use the [InsertItem](#) method to insert a child item.

property ComboBox.IntegralHeight as Boolean

Gets or sets a value indicating whether the control should resize to avoid showing partial items.

Type	Description
Boolean	A boolean expression that indicates whether the control displays partial items.

The `IntegralHeight` property has effect only if the control's [Style](#) property is `DropDown` or `DropDownList`. By default, the `IntegralHeight` property is `False`. Use the [HeightList](#) property to specify the height of the drop down window. Use the [DefaultItemHeight](#) property to specify the default height for the items. Use the `IntegralHeight` property for lists with items that have the same height.

property ComboBox.ItemFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, ColIndex as Long, HitTestInfo as HitTestInfoEnum) as HITEM

Retrieves the item 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.
ColIndex as Long	A long expression that indicates on return, the column where the point belongs.
HitTestInfo as HitTestInfoEnum	A HitTestInfoEnum expression that determines on return the position of the cursor within the cell.
HITEM	A long expression that indicates the column where the point belongs.

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 handle of the item from the cursor.** The ItemFromPoint returns 0 if no item is over the cursor. Use the [ColumnFromPoint](#) property to retrieve the column from point. Use the [SelectableItem](#) property to specify the user can select an item.

The following VB sample prints the caption of the cell over the cursor:

```
Private Sub ComboBox1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        Dim i As HITEM, c As Long, hit As EXCOMBOBOXLibCtl.HitTestInfoEnum
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If (i >= 0) Then
            Debug.Print .Items.CellCaption(i, c) & " HT = " & hit
        End If
    End With
End Sub
```

The following VB sample displays the index of the icon being clicked (use the [CloseOnDoubleClick](#) property to specify whether the user closes the drop down portion of the control by a single or double click):

```
Private Sub ComboBox1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim h As HITEM, c As Long, hit As HitTestInfoEnum
    h = ComboBox1.ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If (Not h = 0) Then
        If exHTCellIcon = (hit And exHTCellIcon) Then
            Debug.Print "The index of icon being clicked is: " & (hit And &HFFFF0000) / 65536
        End If
    End If
End Sub
```

The following C++ sample prints the cell from the caption and the hit test code:

```
void OnMouseMoveCombobox1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, i = m_combobox.GetItemFromPoint( X, Y, &c, &hit );
    if ( i != 0 )
    {
        CString strOutput = V2S( &m_combobox.GetItems().GetCellCaption(COleVariant(i), COleVariant(c) ) );
        strOutput += " HT = ";
        strOutput += V2S(COleVariant(hit));
        strOutput += "\r\n";
        OutputDebugString( strOutput );
    }
}
```

The following VB.NET sample prints the cell from the caption and the hit test code:

```
Private Sub AxComboBox1_MouseMoveEvent(ByVal sender As Object, ByVal e As AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent) Handles AxComboBox1.MouseMoveEvent
    With AxComboBox1
```

```

Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
i = .get_ItemFromPoint(e.x, e.y, c, hit)
If (i >= 0) Then
    Debug.WriteLine(.Items.CellCaption(i, c) & " HT = " & hit.ToString())
End If
End With
End Sub

```

The following C# sample prints the cell from the caption and the hit test code:

```

private void axComboBox1_MouseMoveEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent e)
{
    EXCOMBOBOXLib.HitTestInfoEnum hit;
    int c = 0, i = axComboBox1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i != 0)
        System.Diagnostics.Debug.WriteLine(axComboBox1.Items.get_CellCaption(i,
c).ToString() + " HT = " + hit.ToString() );
}

```

The following VFP sample prints the cell from the caption and the hit test code:

```

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

With thisform.ComboBox1
    local c, hit
    c = 0
    hit = 0
    .Items.DefaultItem = .ItemFromPoint(X , Y, @c, @hit)
    If (.Items.DefaultItem >= 0) Then
        wait window nowait .Items.CellCaption(0, c) + " HT = " + str(hit)
    EndIf
EndWith

```

property ComboBox.Items as Items

Retrieves the control's item collection.

Type	Description
Items	An Items object that holds the control's items collection.

Use Items property to access the control's items collection. Use the [Add](#) method to add new columns to the control. Use the [AddItem](#), [InsertItem](#), [PutItems](#) method to to add new items to the control. Use the [GetItems](#) method to get the items collection into a safe array. Stores the Items property to a member variable, if you are using Items property multiple times, to maintain performance. Use the [BeginUpdate](#) and [EndUpdate](#) to avoid updating the control while adding new items and columns to the control.

For instance, the following VB sample shows how to hold the Items property to a member variable:

```
For i = 0 To 100
    ComboBox1.Items.AddItem "1"
    ComboBox1.Items.AddItem "2"
Next
```

Instead you can use a sample like follows:

```
Dim its As Items
Set its = ComboBox1.Items
For i = 0 To 100
    its.AddItem "1"
    its.AddItem "2"
Next
```

or

```
With ComboBox1.Items
    For i = 0 To 100
        .AddItem "1"
        .AddItem "2"
    Next
End With
```

as well.

The following VB sample adds a column and some items to a drop down list control:

```
With ComboBox1
    .BeginUpdate
    .Style = DropDownList
    .ColumnAutoResize = True
    .Columns.Add "Column 1"
    .PutItems Array(1, "Item 2", 3, 4, 5)
    .Items.SelectItem(.Items.FindItem("Item 2")) = True
    .EndUpdate
End With
```

The following C++ gets the Items collection to a member variable:

```
CItems its = m_combobox.GetItems();
```

The following VB.NET gets the Items collection to a member variable:

```
Dim its As EXCOMBOBOXLib.Items = AxComboBox1.Items
```

or

```
With AxComboBox1.Items

End With
```

as well.

The following C# gets the Items collection to a member variable:

```
EXCOMBOBOXLib.Items its = axComboBox1.Items;
```

The following VFP gets the Items collection to a member variable:

```
With thisform.ComboBox1
EndWith
```

or

```
local its
its = thisform.ComboBox1
```

as well

property ComboBox.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 ComboBox.Key(KeyCode as Integer) as Integer

Replaces a virtual key.

Type	Description
KeyCode as Integer	An integer expression that indicates the key code being replaced.
Integer	An integer expression that indicates the replaced key code.

Use the Key property to replace the default action when a certain key is pressed. Use the [KeyDown](#) event to notify your application that the user presses a key. For instance, Key(vbKeyF2) = vbKeyF4 specifies that F2 and F4 keys do the same thing. By default, the F4 key opens the drop down window, so F2 will open the drop down window too.

The following VB sample disables the F4 key (by default, the F4 key shows or hides the drop down portion of the control):

```
ComboBox1.Key(vbKeyF4) = 0
```

The following VB sample specifies that user may open the drop down window using the F2 key, and can't open the drop down window using the F4 key:

```
With ComboBox1  
    .Key(vbKeyF2) = vbKeyF4  
    .Key(vbKeyF4) = 0  
End With
```

The following C++ sample disables the F4 key (by default, the F4 key shows or hides the drop down portion of the control):

```
m_combobox.SetKey( VK_F4, 0 );
```

The following VB.NET sample disables the F4 key (by default, the F4 key shows or hides the drop down portion of the control):

```
With AxComboBox1  
    .set_Key(Convert.ToInt16(Keys.F4), 0)  
End With
```

The following C# sample disables the F4 key (by default, the F4 key shows or hides the drop down portion of the control):

```
axComboBox1.set_Key(Convert.ToInt16(Keys.F4), 0);
```

The following VFP sample disables the F4 key (by default, the F4 key shows or hides the drop down portion of the control):

```
with thisform.ComboBox1  
    .Key(115) = 0 && vkF4  
endwith
```

property ComboBox.LabelColumnIndex as Long

Specifies a different column (index) to be displayed on the control's label, while the SingleEdit property is True.

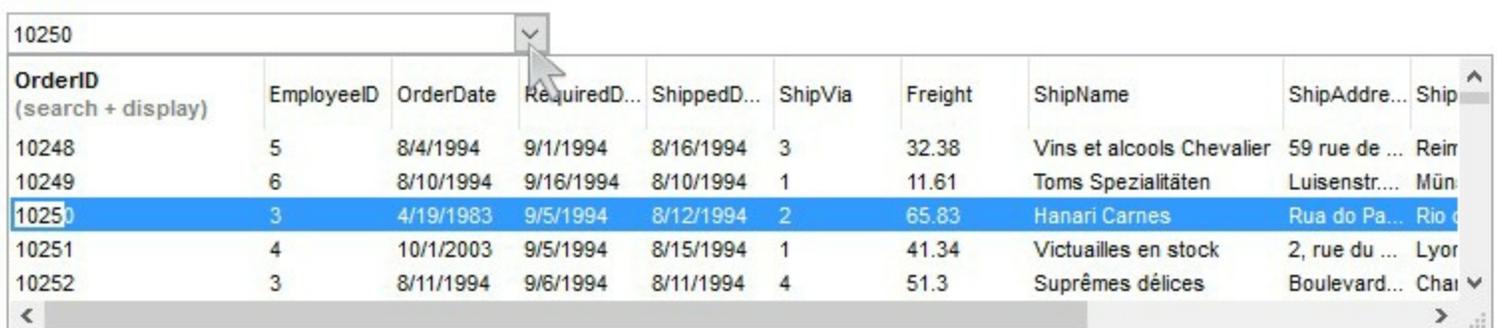
Type	Description
Long	A Long expression that specifies the index of the column to be displayed on the control's label, when the SingleEdit property is True.

By default, the LabelColumnIndex property is -1, which indicates that the [SearchColumnIndex](#) property is used instead. In other words, by default the LabelColumnIndex property has no effect, while it points to a non-existing column. For instance, if the control is hosted as an user editor by eXGrid/eXComboBox control, you can specify a different column to be displayed in place. Use the [LabelText](#) property to display a custom label.

The LabelColumnIndex property has effect only if:

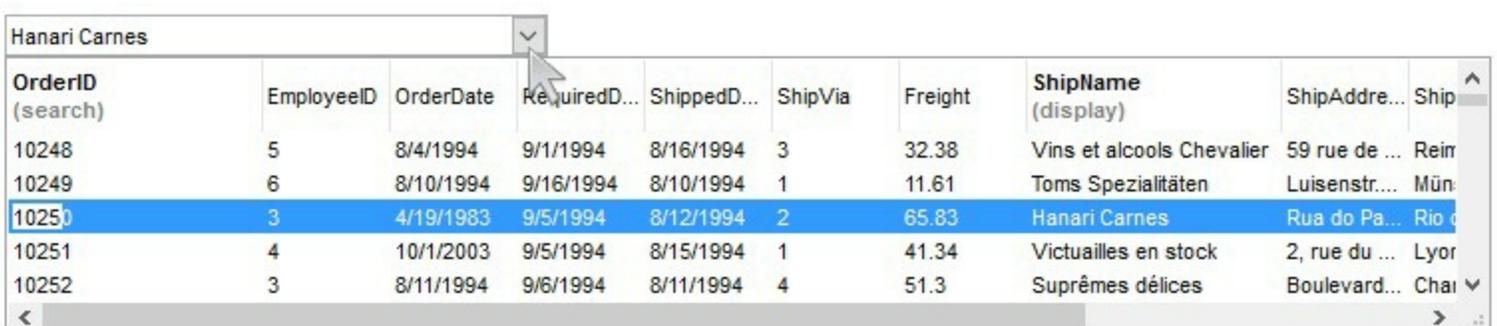
- [Style](#) property is DropDownList
- [SingleEdit](#) property is True

The following screen shot shows the control with SearchColumnIndex property on **0** (by default), and LabelColumnIndex property on **-1** (by default):



OrderID (search + display)	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia	Freight	ShipName	ShipAdresse...	Ship
10248	5	8/4/1994	9/1/1994	8/16/1994	3	32.38	Vins et alcools Chevalier	59 rue de ...	Rein
10249	6	8/10/1994	9/16/1994	8/10/1994	1	11.61	Toms Spezialitäten	Luisenstr...	Mün
10250	3	4/19/1983	9/5/1994	8/12/1994	2	65.83	Hanari Carnes	Rua do Pa...	Rio
10251	4	10/1/2003	9/5/1994	8/15/1994	1	41.34	Victuailles en stock	2, rue du ...	Lyor
10252	3	8/11/1994	9/6/1994	8/11/1994	4	51.3	Suprêmes délices	Boulevard...	Char

The following screen shot shows the control with SearchColumnIndex property on **0** (by default), and LabelColumnIndex property on **7**:



OrderID (search)	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia	Freight	ShipName (display)	ShipAdresse...	Ship
10248	5	8/4/1994	9/1/1994	8/16/1994	3	32.38	Vins et alcools Chevalier	59 rue de ...	Rein
10249	6	8/10/1994	9/16/1994	8/10/1994	1	11.61	Toms Spezialitäten	Luisenstr...	Mün
10250	3	4/19/1983	9/5/1994	8/12/1994	2	65.83	Hanari Carnes	Rua do Pa...	Rio
10251	4	10/1/2003	9/5/1994	8/15/1994	1	41.34	Victuailles en stock	2, rue du ...	Lyor
10252	3	8/11/1994	9/6/1994	8/11/1994	4	51.3	Suprêmes délices	Boulevard...	Char

The following screen shot shows the control with SearchColumnIndex property on **-1**,

LabelColumnIndex property on **-1**, AdjustSearchColumn property on **False**, LabelText property on "**this is just a custom label**":

this is just a custom label									
OrderID	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia	Freight	ShipName	ShipAddre...	Ship
10248	5	8/4/1994	9/1/1994	8/16/1994	3	32.38	Vins et alcools Chevalier	59 rue de ...	Rein
10249	6	8/10/1994	9/16/1994	8/10/1994	1	11.61	Toms Spezialitäten	Luisenstr...	Mün
10250	3	4/19/1983	9/5/1994	8/12/1994	2	65.83	Hanari Carnes	Rua do Pa...	Rio t
10251	4	10/1/2003	9/5/1994	8/15/1994	1	41.34	Victuailles en stock	2, rue du ...	Lyor
10252	3	8/11/1994	9/6/1994	8/11/1994	4	51.3	Suprêmes délices	Boulevard...	Char

property ComboBox.LabelHeight as Long

Retrieves or sets a value that indicates the label's height, in pixels.

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

The LabelHeight property defines the height of the control's label in pixels. By default, the LabelHeight property is 21 pixels. Use the LabelHeight property to define the height for the control's label. Use the [HeaderHeight](#) property to specify the height of the control's header bar. Use the [HeightList](#) and [MinHeightList](#) properties to specify at runtime the height and the minimum height of the drop down portion of the control. The control's label area display the control's drop down button. Use the [SingleEdit](#) property to specify whether a single or multiple columns are displayed on the control's label area. The [LabelColumnIndex](#) property specifies a different column (index) to be displayed on the control's label, while the [SingleEdit](#) property is True.

property ComboBox.LabelText as String

Specifies the HTML caption to be displayed in the control's label when SearchColumnIndex property points to a not-existing column, AdjustSearchColumn property in False, the SingleEdit property is True, and the Style property is DropDownList.

Type	Description
String	A String expression that specifies the HTML caption being displayed on the control's label.

By default, the LabelText property is empty. Use the LabelText property to specify the HTML caption being displayed on the control's label. Shortly, the LabelText property allows you to specify a custom text for the control's label no matter of what the user selected in the drop down portion of the control.

The LabelText property has effect only if:

- [Style](#) property is DropDownList
- [SingleEdit](#) property is True

If:

- [SingleSel](#) property is False (multiple-selection)
- [LabelColumnIndex](#) property points to a valid column
- [SearchColumnIndex](#) property points to a valid column

the LabelText property specifies the HTML format to display the count of selected items By default (if empty), the format is "(%c)" which displays the number of selected items between () parenthesis. For instance, "<bgcolor=000000><fgcolor=FFFFFF> %c </fgcolor></bgcolor>" shows the count of selected items in white on a black-background. The " " prevents showing the count of selected items

The following screen show shows the control's label (multiple-selection)



If:

- [SingleSel](#) property is true (single-selection)
- [SearchColumnIndex](#) property points to a not-existing column (for instance -1)

- [LabelColumnIndex](#) property points to a not-existing column (for instance -1)
- [AdjustSearchColumn](#) property in False

the LabelText property specifies the HTML caption to show within the control's label (for instance, displays the list of checked-items, as soon as user changes the cell's state)

The following screen shot shows the control's label displaying the list of checked items:



The following VB sample changes the LabelText property once a check box state is changed, and allow checking a cell once the user clicks an item:

```
Private Sub ComboBox1_CellStateChanged(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    Dim s As String
    With ComboBox1
        For Each i In .Items
            With .Items
                If Not (.CellState(i, 0) = 0) Then
                    s = s + If(Len(s) = 0, "", ",") & .CellCaption(i, 0)
                End If
            End With
        End With
    Next
    .LabelText = " " & s & " "
End With
End Sub
```

```
Private Sub ComboBox1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        Dim h As Long, c As Long, hit As HitTestInfoEnum
        h = .ItemFromPoint(-1, -1, c, hit)
        If Not (exHTCellCheck = (hit And exHTCellCheck)) Then
            With .Items
                .CellState(.FocusItem, 0) = (1 + .CellState(.FocusItem, 0)) Mod 2
            End With
        End With
    End With
```

End If
End With
End Sub

The following VB/NET sample changes the LabelText property once a check box state is changed, and allow checking a cell once the user clicks an item:

```
private void excombobox1_CellStateChanged(object sender, int Item, int ColIndex)
{
    string s = "";
    exontrol.EXCOMBOBOXLib.Items items = excombobox1.Items;
    for ( int i = 0; i < items.ItemCount; i++ )
    {
        int h = items.get_ItemByIndex(i);
        if ( items.get_CellState(h,0) != 0 )
            s = s + (s.Length == 0 ? "": ",") + items.get_CellCaption(h, 0);
    }
    excombobox1.LabelText = " " + s + " ";
}

private void excombobox1_MouseUpEvent(object sender, short Button, short Shift, int X,
int Y)
{
    int c = 0;
    exontrol.EXCOMBOBOXLib.HitTestInfoEnum hit =
exontrol.EXCOMBOBOXLib.HitTestInfoEnum.exHTCell;
    int h = excombobox1.get_ItemFromPoint(-1, -1, ref c, ref hit);
    if (exontrol.EXCOMBOBOXLib.HitTestInfoEnum.exHTCellCheck != (hit &
exontrol.EXCOMBOBOXLib.HitTestInfoEnum.exHTCellCheck))
    {
        exontrol.EXCOMBOBOXLib.Items items = excombobox1.Items;
        items.set_CellState(items.FocusItem, 0, (1 + items.get_CellState(items.FocusItem, 0)) %
2);
    }
}
```

The LabelText property supports the following HTML tags:

- ` ... ` displays the text in **bold**
- `<i> ... </i>` displays the text in *italics*
- `<u> ... </u>` underlines the text
- `<s> ... </s>` Strike-through text
- `<a id;options> ... ` 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=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **number[:width]** 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.
- **key[:width]** 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 **&**; (&), **<**; (<), **>**; (>), **"**; (") and **&#number;**; (the character with specified code), For instance, the **€** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **** to define a smaller or a larger font to be displayed. For instance: "Text with **<off 6>**subscript" displays the text such as: Text with subscript The "Text with **<off -6>**superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a

value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<gra FFFFFFFF;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 ComboBox.LinesAtRoot as LinesAtRootEnum

Link items at the root of the hierarchy.

Type	Description
LinesAtRootEnum	A LinesAtRootEnum expression that indicates whether the control links the items at the root of the hierarchy.

The control paints the hierarchy lines to the right if the Column's [Alignment](#) property is RightAlignment. The [TreeColumnIndex](#) property specifies the index of column where the hierarchy lines are painted. Use the [Indent](#) property to increase or decrease the amount, in pixels, that child items are indented relative to their parent items. Use the [HasLines](#) property to enhances the graphic representation of a tree control's hierarchy by drawing lines that link child items to their corresponding parent item. Use the [InsertItem](#) method to insert a child item.

property ComboBox.Locked as Boolean

Determines whether a control can be edited.

Type	Description
Boolean	A boolean expression that determines whether a control can be edited.

Use the Locked property to lock the control. When the control is locked the user is able to drop the control's list but he cannot change the selected value. Use the [Enabled](#) property to disable the control. Use the Locked property to refresh the font in the edit controls, after changing the control's font. Use the [CellEnabled](#) property to disable a cell. Use the [EnableItem](#) property to disable the item. For instance, changing the Locked property recreates the edit controls in the control's label area, if the [Style](#) property is DropDown. If the control is locked, the edit controls in the control's label are read only. The system handles the look and appearance for a read only edit control.

The following VB sample changes the control's font at runtime:

```
With ComboBox1.Font
    .Name = "Comic Sans MS"
    .Bold = True
End With
ComboBox1.Locked = ComboBox1.Locked
```

The following C++ sample changes the control's font at runtime:

```
#include "font.h"
COleFont font = m_combobox.GetFont();
font.SetName("Comic Sans MS");
font.SetBold( TRUE );
m_combobox.SetLocked( m_combobox.GetLocked() );
```

The following VB.NET sample changes the control's font at runtime:

```
AxComboBox1.Font = New System.Drawing.Font("Comic Sans MS", 10, FontStyle.Bold)
AxComboBox1.Locked = AxComboBox1.Locked
```

The following C# sample changes the control's font at runtime:

```
axComboBox1.Font = new System.Drawing.Font("Comic Sans MS", 10, FontStyle.Bold);
```

```
axComboBox1.Locked = axComboBox1.Locked;
```

The following VFP sample changes the control's font at runtime:

```
With thisform.ComboBox1.Font
```

```
  .Name = "Comic Sans MS"
```

```
  .Bold = .t.
```

```
EndWith
```

```
thisform.ComboBox1.Locked = thisform.ComboBox1.Locked
```

property ComboBox.MarkSearchColumn as Boolean

Retrieves or sets a value that indicates whether the searching column is marked or unmarked

Type	Description
Boolean	A boolean value that indicates whether the control marks the searching column.

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 or 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.

property ComboBox.MinHeightList as Long

Retrieves or sets a value that indicates the minimum height of the control's drop down window.

Type	Description
Long	A long expression that indicates the minimum height of the control's drop down window.

Use the `MinHeightList` property to specify the minimum height of the control's drop down window. The property has effect only if the control's style is `DropDown` or `DropDownList`. Use the [HeightList](#) property to specify the height of the control's drop down window. Use the [AllowSizeGrip](#) property to specify whether the drop down portion of the control is resizable at runtime. Use the [AllowHResize](#), [AllowVResize](#) properties to specify whether the width, height of the drop down portion of the control are resizable.

property ComboBox.MinWidthList as Long

Retrieves or sets a value that indicates the minimum width of the control's drop down window.

Type	Description
Long	A long expression that indicates the minimum width of the control's drop down window.

Use the `MinHeightList` property to specify the minimum width of the control's drop down window. The property has effect only if the control's style is `DropDown` or `DropDownList`. Use the [WidthList](#) property to change the width of the control's drop down window. Use the [Alignment](#) property to change the drop down list's alignment. Use the [AllowSizeGrip](#) property to specify whether the drop down portion of the control is resizable at runtime. Use the [AllowHResize](#), [AllowVResize](#) properties to specify whether the width, height of the drop down portion of the control are resizable.

method ComboBox.PutItems (Items as Variant, [Parent as Variant])

Adds an array of integer, long, date, string, double, float, or variant arrays to the list.

Type	Description
Items as Variant	A safe array that control uses to fill with.
Parent as Variant	A long expression that specifies the handle of the item where the array is being inserted, or 0 if missing

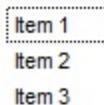
The PutItems method loads items from a safe array. The Parent parameter of the PutItems method specifies the handle of the item where the array is being inserted as child items. Use the [AddItem](#) method to add a single item to the control. Use the [InsertItem](#) method to insert a child item to the control. Use the [ItemPosition](#) property to specify the item's position. Use the [GetItems](#) method to get the items collection to a safe array. Use the [ColumnAutoResize](#) property to specify whether the visible columns should fit the control's client area. 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 6 sample loads a flat array to a single column control (and shows as in the following picture):

```
With ComboBox1
    .BeginUpdate
    .Columns.Add "Column 1"
    .PutItems Array("Item 1", "Item 2", "Item 3")
    .EndUpdate
End With
```

or similar for /NET Assembly version:

```
With Excombobox1
    .BeginUpdate()
    .Columns.Add("Column 1")
    .PutItems(New String() {"Item 1", "Item 2", "Item 3"})
    .EndUpdate()
End With
```



Item 1
Item 2
Item 3

The following VB 6 sample loads a hierarchy to a single column control (and shows as in the following picture):

With ComboBox1

```
.BeginUpdate
```

```
.LinesAtRoot = exLinesAtRoot
```

```
.Columns.Add ""
```

```
.PutItems Array("Root 1", Array("Child 1.1", Array("Sub Child 1.1.1", "Sub Child 1.1.2"),  
"Child 1.2"), "Root 2", Array("Child 2.1", "Child 2.2"))
```

```
.EndUpdate
```

```
End With
```

or similar for /NET Assembly version:

With Excombobox1

```
.BeginUpdate()
```

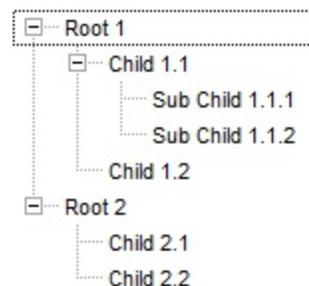
```
.LinesAtRoot = excontrol.EXCOMBOBOXLib.LinesAtRootEnum.exLinesAtRoot
```

```
.Columns.Add("")
```

```
.PutItems(New Object() {"Root 1", New Object() {"Child 1.1", New String() {"Sub Child  
1.1.1", "Sub Child 1.1.2"}, "Child 1.2"}, "Root 2", New String() {"Child 2.1", "Child 2.2"}})
```

```
.EndUpdate()
```

```
End With
```



The following VB 6 sample loads a list of items, in a three columns control (as shown in the following picture):

```
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"
```

With ComboBox1

```
.BeginUpdate  
.Columns.Add "Column 1"  
.Columns.Add "Column 2"  
.Columns.Add "Column 3"  
  
.PutItems v  
.EndUpdate
```

End With

Column 1	Column 2	Column 3
One	One	One
Two	Two	Two
Three	Three	Three

The following VB 6 sample loads a list of items, in a three columns control (as shown in the following picture):

```
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"
```

With ComboBox1

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

```
.Items.AddItem "Root"
```

```
.PutItems v, .Items.FirstVisibleItem
```

```
.EndUpdate
```

```
End With
```

Column 1	Column 2	Column 3
[-] Root		
One	One	One
Two	Two	Two
Three	Three	Three

The following VB sample shows how to load 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
```

```
ComboBox1.BeginUpdate  
For Each f In rs.Fields  
    ComboBox1.Columns.Add f.Name  
Next  
ComboBox1.PutItems rs.GetRows()  
ComboBox1.EndUpdate
```

The following VB.NET sample adds a column and some items to a drop down list control:

```
With AxComboBox1  
    .BeginUpdate()  
    .ColumnAutoSize = True  
    .Columns.Add("Column 1")  
    Dim o() As Object = {1, "Item 2", 3, 4, 5}  
    .PutItems(o)  
    .Items.SelectItem(.Items.FindItem("Item 2")) = True  
    .EndUpdate()  
End With
```

property ComboBox.RadiolImage(Checked as Boolean) as Long

Retrieves or sets a value that indicates the image used by cells of radio type.

Type	Description
Checked as Boolean	A boolean expression that indicates the radio's state. True means checked, and False means unchecked.
Long	A long expression that indicates the index of image used to paint the radio button. 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 RadiolImage and [CheckImage](#) properties to define the icons used for radio and check box cells. The RadiolImage 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 following VB sample changes the default icon for the cells of radio type:

```
ComboBox1.RadiolImage(True) = 1      ' Sets the icon for cells of radio type that are checked
ComboBox1.RadiolImage(False) = 2    ' Sets the icon for cells of radio type that are unchecked
```

The ComboBox1.RadiolImage(True) = 0 makes the control to use the default icon for painting cells of radio type that are checked.

The following C++ sample changes the default icon for the cells of radio type:

```
m_combobox.SetRadiolImage( TRUE, 1 );
m_combobox.SetRadiolImage( FALSE, 2 );
```

The following VB.NET sample changes the default icon for the cells of radio type:

```
With AxComboBox1
```

```
    .set_RadiolImage(True, 1)
```

```
    .set_RadiolImage(False, 2)
```

```
End With
```

The following C# sample changes the default icon for the cells of radio type:

```
axComboBox1.set_RadiolImage(true, 1);
```

```
axComboBox1.set_RadiolImage(false, 2);
```

The following VFP sample changes the default icon for the cells of radio type:

```
with thisform.ComboBox1
```

```
    local sT, sCR
```

```
    sCR = chr(13) + chr(10)
```

```
    sT = "RadiolImage(True) = 1"+ sCR
```

```
    sT = sT + "RadiolImage(False) = 2"+ sCR
```

```
    .Template = sT
```

```
endwith
```

The VFP considers the RadiolImage call as being a call for an array, so an error occurs if the method is called directly, so we built a template string that we pass to the [Template](#) property

method **ComboBox.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:

```
ComboBox1.Refresh
```

The following C++ sample calls the Refresh method:

```
m_combobox.Refresh();
```

The following VB.NET sample calls the Refresh method:

```
AxComboBox1.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:

```
axComboBox1.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.ComboBox1.Object.Refresh()
```

method `ComboBox.RemoveSelection ()`

Removes the selected items (including the descendents)

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

The `RemoveSelection` method removes the selected items (including the descendents). The [RemoveItem](#) method removes a specific item (if it has no child items). The [UnselectAll](#) method unselects all items.

method **ComboBox.Replacelcon** ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the control's image list.

Type	Description
Icon as Variant	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 `Replacelcon` property to add, remove or replace an icon in the control's images collection. Also, the `Replacelcon` property can clear the images collection. Use the [Images](#) method to attach a image list to the control. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection.

The following sample shows how to add a new icon to control's images list:

```
i = ExComboBox1.Replacelcon( 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 = ExComboBox1.Replacelcon( 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:

```
ExComboBox1.Replacelcon 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:

```
ExComboBox1.Replacelcon 0, -1
```

property ComboBox.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 RightToLeft property affects all columns, and future columns being added. Use the [Alignment](#) property to align all the cells in the column. Use the [CellAlignment](#) property to align a particular cell. Use the [HeaderAlignment](#) property to align the column's caption in the control's header. Use the [EditAlignment](#) property to align the text inside the edit control, if present in the control's label. Use the [Alignment](#) property to align the drop down portion of the control relative to the control's label.

Changing the RightToLeft property on True does the following:

- displays the vertical scroll bar on the left side of the control
- 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 following screen shot shows the control when RightToLeft property is True (the red areas mark the new position for elements like drop down, vertical scroll bar, hierarchy lines):



(By default) Changing the RightToLeft property on False does the following:

- displays the vertical scroll bar on the right side of the control
- 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 following screen shot shows the control when RightToLeft property is False (by default):



property ComboBox.ScrollButtonHeight as Long

Specifies the height of the button in the vertical scrollbar.

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

By default, the ScrollButtonHeight property is -1. If the ScrollButtonHeight property is -1, the control uses the default height (from the system) for the buttons in the vertical scroll bar. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb.

property ComboBox.ScrollButtonWidth as Long

Specifies the width of the button in the horizontal scrollbar.

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

By default, the ScrollButtonWidth property is -1. If the ScrollButtonWidth property is -1, the control uses the default width (from the system) for the buttons in the horizontal scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb.

property ComboBox.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 recommended 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 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 [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 ComboBox.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 [ScrollPartCaption](#) 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 [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. The control fires the [ScrollButtonClick](#) event when the user clicks a part of the scroll bar.

property ComboBox.ScrollHeight as Long

Specifies the height of the horizontal scrollbar.

Type	Description
Long	A long expression that defines the height of the horizontal scroll bar.

By default, the ScrollHeight property is -1. If the ScrollHeight property is -1, the control uses the default height of the horizontal scroll bar from the system. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb.

property ComboBox.ScrollOnDrop as Boolean

Specifies a value that indicates whether the drop down list is scrolled when it is shown.

Type	Description
Boolean	A boolean expression that indicates whether the drop down portion of the control is scrolled when it is shown.

By default, the ScrollOnDrop property is True. The ScrollOnDrop property has effect only if the [Style](#) property is DropDown or DropDownList. If the ScrollOnDrop property is False, the control shows directly the drop down portion of the control without scrolling it.

property ComboBox.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 fourth 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 fourth 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.

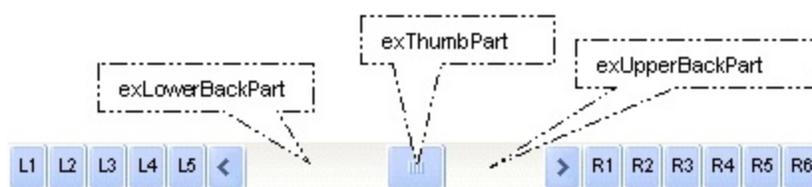
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property ComboBox.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 displayed
String	A String expression that specifies the caption being displayed on the part of the scroll bar.

Use the ScrollPartCaption 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 [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. 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 :

```
.BeginUpdate
```

```
.ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
```

```
.ScrollPartCaption(exVScroll, exLeftB1Part) = "<img> </img> 1"
```

```
.ScrollPartCaption(exVScroll, exRightB1Part) = "<img> </img> 2"
```

```
.EndUpdate
```

```
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxComboBox1
```

```
.BeginUpdate()
```

```
.set_ScrollPartVisible(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part Or  
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, True)
```

```
.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part, "<img> </img> 1")
```

```
.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.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 :

```
axComboBox1.BeginUpdate();
```

```
axComboBox1.set_ScrollPartVisible(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part |  
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, true);
```

```
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part , "<img> </img> 1");
```

```
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, "<img> </img> 2");
```

```
axComboBox1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_comboBox.BeginUpdate();
```

```
m_comboBox.SetScrollPartVisible( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_comboBox.SetScrollPartCaption( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ , _T("<img>
</img> 1" ) );
m_comboBox.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img>
</img> 2" ) );
m_comboBox.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.ComboBox1
  .BeginUpdate
    .ScrollPartVisible(0, bitor(32768,32)) = .t.
    .ScrollPartCaption(0,32768) = "<img> </img> 1"
    .ScrollPartCaption(0, 32) = "<img> </img> 2"
  .EndUpdate
EndWith
```

*** ActiveX Control Event ***

LPARAMETERS scrollpart

```
wait window nowait ltrim(str(scrollpart))
```

property `ComboBox.ScrollPartCaptionAlignment`(`ScrollBar` as `ScrollBarEnum`, `Part` as `ScrollPartEnum`) as `AlignmentEnum`

Specifies the alignment of the caption in the part of the scroll bar.

Type	Description
<code>ScrollBar</code> as ScrollBarEnum	A <code>ScrollBar</code> expression that indicates the scrollbar where the caption is displayed.
<code>Part</code> as ScrollPartEnum	A <code>ScrollPartEnum</code> expression that specifies the parts of the scroll where the text is displayed
AlignmentEnum	An <code>AlignmentEnum</code> 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 [ScrollPartCaption](#) 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 ComboBox1
    .MinHeightList = 304
    .ColumnAutoResize = True
    .MinHeightList = 304
    .ColumnAutoResize = True
    .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
    .Columns.Add 5
    .Columns.Add 6
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 AxComboBox1
    .MinHeightList = 304
    .ColumnAutoResize = True
    .MinHeightList = 304
    .ColumnAutoResize = True

    .set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,EXCOMBOBOXLib.ScrollF

    .set_ScrollPartCaptionAlignment(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,EXCOMBOBOX

    .set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,EXCOMBOBOXLib.ScrollF

    .set_ScrollPartCaptionAlignment(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,EXCOMBOBOX

    .ColumnAutoResize = False
    .Columns.Add 1
    .Columns.Add 2
    .Columns.Add 3
    .Columns.Add 4
    .Columns.Add 5
    .Columns.Add 6
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:

```
axComboBox1.MinHeightList = 304;
axComboBox1.ColumnAutoResize = true;
axComboBox1.MinHeightList = 304;
axComboBox1.ColumnAutoResize = true;
```

```

axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,EXCOMBO
axComboBox1.set_ScrollPartCaptionAlignment(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,EXCOMBO
axComboBox1.set_ScrollPartCaptionAlignment(EXCOMBOBOXLib.ScrollBarEnum.exHScroll,

axComboBox1.ColumnAutoResize = false;
axComboBox1.Columns.Add(1.ToString());
axComboBox1.Columns.Add(2.ToString());
axComboBox1.Columns.Add(3.ToString());
axComboBox1.Columns.Add(4.ToString());
axComboBox1.Columns.Add(5.ToString());
axComboBox1.Columns.Add(6.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 'EXCOMBOBOXLib' for the library: 'ExComboBox 1.0 Control
    Library'

    #import "D:\\Exontrol\\ExComboBox\\project\\Demo\\ExComboBox.dll"
    using namespace EXCOMBOBOXLib;
*/
EXCOMBOBOXLib::IComboBoxPtr spComboBox1 = GetDlgItem(IDC_COMBOBOX1)-
>GetControlUnknown();
spComboBox1->PutMinHeightList(304);
spComboBox1->PutColumnAutoResize(VARIANT_TRUE);
spComboBox1->PutMinHeightList(304);
spComboBox1->PutColumnAutoResize(VARIANT_TRUE);
spComboBox1-
>PutScrollPartCaption(EXCOMBOBOXLib::exHScroll,EXCOMBOBOXLib::exLowerBackPart,L"l

spComboBox1-

```

```
> PutScrollPartCaptionAlignment(EXCOMBOBOXLib::exHScroll,EXCOMBOBOXLib::exLowerB
```

```
spComboBox1-
```

```
> PutScrollPartCaption(EXCOMBOBOXLib::exHScroll,EXCOMBOBOXLib::exUpperBackPart,L"r
```

```
spComboBox1-
```

```
> PutScrollPartCaptionAlignment(EXCOMBOBOXLib::exHScroll,EXCOMBOBOXLib::exUpperB
```

```
spComboBox1->PutColumnAutoResize(VARIANT_FALSE);
```

```
spComboBox1->GetColumns()->Add(L"1");
```

```
spComboBox1->GetColumns()->Add(L"2");
```

```
spComboBox1->GetColumns()->Add(L"3");
```

```
spComboBox1->GetColumns()->Add(L"4");
```

```
spComboBox1->GetColumns()->Add(L"5");
```

```
spComboBox1->GetColumns()->Add(L"6");
```

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.ComboBox1
```

```
  .MinHeightList = 304
```

```
  .ColumnAutoResize = .T.
```

```
  .MinHeightList = 304
```

```
  .ColumnAutoResize = .T.
```

```
  .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)
```

```
  .Columns.Add(5)
```

```
  .Columns.Add(6)
```

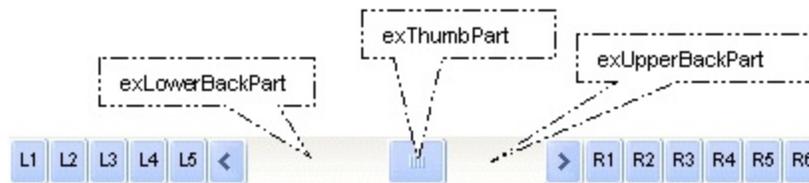
```
endwith
```

property `ComboBox.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 [ScrollPartCaption](#) 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. 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.

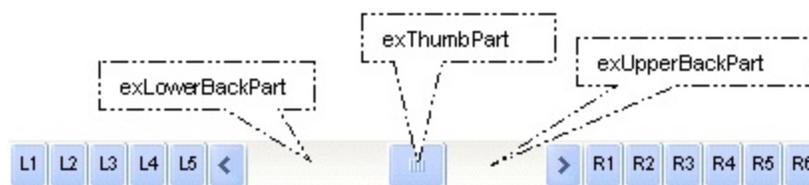


property ComboBox.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

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 [ScrollPartCaption](#) 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. 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 :

```
.BeginUpdate
```

```
.ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
```

```
.ScrollPartCaption(exVScroll, exLeftB1Part) = "<img> </img> 1"
```

```
.ScrollPartCaption(exVScroll, exRightB1Part) = "<img> </img> 2"
```

```
.EndUpdate
```

```
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxComboBox1
```

```
.BeginUpdate()
```

```
.set_ScrollPartVisible(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part Or  
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, True)
```

```
.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part, "<img> </img> 1")
```

```
.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.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 :

```
axComboBox1.BeginUpdate();
```

```
axComboBox1.set_ScrollPartVisible(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part |  
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, true);
```

```
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part , "<img> </img> 1");
```

```
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,  
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, "<img> </img> 2");
```

```
axComboBox1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_comboBox.BeginUpdate();
```

```
m_comboBox.SetScrollPartVisible( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_comboBox.SetScrollPartCaption( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ , _T("<img>
</img> 1" ) );
m_comboBox.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img>
</img> 2" ) );
m_comboBox.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.ComboBox1
  .BeginUpdate
    .ScrollPartVisible(0, bitor(32768,32)) = .t.
    .ScrollPartCaption(0,32768) = "<img> </img> 1"
    .ScrollPartCaption(0, 32) = "<img> </img> 2"
  .EndUpdate
EndWith
```

*** ActiveX Control Event ***

LPARAMETERS scrollpart

```
wait window nowait ltrim(str(scrollpart))
```

property ComboBox.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\(exVSTThumb\)](#) or [Background\(exHSTThumb\)](#) 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 ComboBox.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.

The following VB sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
Private Sub ComboBox1_OffsetChanged(ByVal Horizontal As Boolean, ByVal NewVal As Long)
    If (Not Horizontal) Then
        ComboBox1.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 AxComboBox1_OffsetChanged(ByVal sender As System.Object, ByVal e As AxEXCOMBOBOXLib._IComboBoxEvents_OffsetChangedEvent) Handles AxComboBox1.OffsetChanged
    If (Not e.horizontal) Then
        AxComboBox1.set_ScrollToolTip(EXCOMBOBOXLib.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 OnOffsetChangedComboBox1(BOOL Horizontal, long NewVal)
{
    if ( !Horizontal )
    {
        CString strFormat;
        strFormat.Format( _T("%i"), NewVal );
        m_comboBox.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 axComboBox1_OffsetChanged(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_OffsetChangedEvent e)
{
    if ( !e.horizontal )
        axComboBox1.set_ScrollToolTip(EXCOMBOBOXLib.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.ComboBox1.ScrollToolTip(0) = "Record " + ltrim(str(newval))
EndIf
```

property ComboBox.ScrollWidth as Long

Specifies the width of the vertical scrollbar.

Type	Description
Long	A long expression that defines the width of the vertical scroll bar.

By default, the ScrollWidth property is -1. If the ScrollWidth property is -1, the control uses the default width of the vertical scroll bar from the system. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb. Use the [HideDropDownButton](#) property to hide the control's drop down button when the control loses the focus. The [DropDownButtonWidth](#) property retrieves or sets the width, in pixels, to display the drop down button of the control.

property ComboBox.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 that indicates the column's index that's used in the incremental searching feature.

When [SingleEdit](#) is False, the number of edit controls is equal with the number of columns. The SearchColumnIndex value should be contained by [0..Columns.Count-1]. The SearchColumnIndex indicates the index of edit control that is focused. The searching column can be changed by the user by pressing TAB or Shift+TAB. To disable changing the searching column by the end user use [UseTabKey](#) property. Use the [AdjustSearchColumn](#) property to allow displaying a hidden column in the control's label area. Use the [Value](#) property to get the selected value, in the column that pointed by the SearchColumnIndex property. Use the [Select](#) property to get or set the selected value on a specified column. Use the [MarkSearchColumn](#) property to hide the rectangle around the searching column. The [LabelColumnIndex](#) property specifies a different column (index) to be displayed on the control's label, while the [SingleEdit](#) property is True.

property ComboBox.SelBackColor as Color

Retrieves or sets a value that indicates the selection background color.

Type	Description
Color	A color expression that indicates the selection 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 SelBackColor and [SelForeColor](#) properties to define the colors that highlight the selected items. The control highlights the selected item only if the SelBackColor and [BackColor](#) properties have different values, and the SelForeColor and [ForeColor](#) properties have different values. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectItem](#), [Select](#) or [Value](#) property to select an item. The [FullRowSelect](#) property specifies whether the selection spans the entire width of the control. [How do I assign a new look for the selected item?](#)

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

```
With ComboBox1
  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_combobox.GetVisualAppearance().Add( 0x23,
```

```
COleVariant(_T("D:\\Temp\\ExComboBox_Help\\selected.ebn")) );  
m_combobox.SetSelBackColor( 0x23000000 );  
m_combobox.SetSelForeColor( 0 );
```

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

```
With AxComboBox1  
  With .VisualAppearance  
    .Add(&H23, "D:\\Temp\\ExComboBox_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):

```
axComboBox1.VisualAppearance.Add(0x23,  
"D:\\Temp\\ExComboBox_Help\\selected.ebn");  
axComboBox1.Template = "SelBackColor = 587202560";
```

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

```
With thisform.ComboBox1  
  With .VisualAppearance  
    .Add(35, "D:\\Temp\\ExComboBox_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. Please check the control's help file for the Add method of the Appearance object. There you will

find almost everything you need to change the visual appearance for most of the UI parts of the control. 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 ComboBox1
```

```
    .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 ComboBox1
```

```
    .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 &H34000000, 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 ComboBox.Select(ColIndex as Variant) as Variant

Finds and selects an item given data on the column.

Type	Description
ColIndex as Variant	A long expression that defines the column's index, or a string expression that defines the column's key or caption.
Variant	A variant value being searched.

Use the `Select` property to select a value in the control, if it contains multiple columns. The control selects only selectable items. The [SelectableItem](#) property specifies whether the user can select an item. The control looks for the value in the given column. If the value is found in the column, no error is fired. Use the [Value](#) property to select a value in a single column control. The [SelectCount](#) property specifies if the control has selected items. Use the [SelectedItem](#) property to select an item given its handle. The [SelectionChanged](#) event is fired when user changes the control's selection. Use the [ItemByIndex](#) property to access an item given its index.

The following VB sample selects the "Cell 4" value on the second column:

```
With ComboBox1
    .BeginUpdate
        .HeaderVisible = False
        .ColumnAutoResize = True
        .Columns.Add "Column 1"
        .Columns.Add "Column 2"
        With .Items
            Dim h As HITEM
            h = .AddItem("Cell 1")
            .CellCaption(h, 1) = "Cell 2"
            h = .AddItem("Cell 3")
            .CellCaption(h, 1) = "Cell 4"
        End With
        .Select("Column 2") = "Cell 4"
    .EndUpdate
End With
```

The following VB sample displays the selected item, using the `Select` property:

```
With ComboBox1
```

```
Debug.Print .Select(.SearchColumnIndex)
End With
```

The following C++ sample displays the selected item, using the Select property:

```
OutputDebugString( m_combobox.GetEditText(
COleVariant(m_combobox.GetSearchColumnIndex() ) ) );
```

The following VB.NET sample displays the selected item, using the Select property:

```
With AxComboBox1
    Debug.WriteLine(.get_Select(.SearchColumnIndex))
End With
```

The following C# sample displays the selected item, using the Select property:

```
System.Diagnostics.Debug.WriteLine(axComboBox1.get_Select(axComboBox1.SearchColumnIndex));
```

The following VFP sample displays the selected item, using the Select property:

```
With thisform.ComboBox1
    wait window nowait .Object.Select(.SearchColumnIndex())
EndWith
```

property ComboBox.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 ComboBox.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.

The SelForeColor and [SelBackColor](#) properties specify the colors to mark the selected items. The control highlights the selected item only if the SelBackColor and [BackColor](#) properties have different values, and the SelForeColor and [ForeColor](#) properties have different values. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectItem](#), [Select](#) or [Value](#) property to select an item. The [FullRowSelect](#) property specifies whether the selection spans the entire width of the control.

property ComboBox.SelLength as Long

Returns or sets the number of characters selected.

Type	Description
Long	A long expression that indicates the number of characters selected

Use the SelLength property to specify the number of characters being selected in the focused edit control. The [SelStart](#) property returns or sets the starting point of text selected. Use the [EditText](#) property to get the text on the control's label. The [SearchColumnIndex](#) property indicates the index of the column that has the focus. The SelLength and SelStart properties have effect only if the [Style](#) property is Simple or DropDown.

property ComboBox.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. Use the [SelLength](#) property to specify the number of characters being selected in the focused edit control. Use the [EditText](#) property to get the text on the control's label. The [SearchColumnIndex](#) property indicates the index of the column that has the focus. The SelLength and SelStart properties have effect only if the [Style](#) property is Simple or DropDown.

property ComboBox.ShowClearButton as Long

Shows or hides the control's clear-button.

Type	Description
Long	A long expression that specifies whether the clear-button is visible (any non-zero value) or hidden (zero) as explained below

By default, the ShowClearButton property is 0 (that indicates that the clear-button is hidden). The ShowClearButton property shows or hides the control's clear-button. The clear button clears the control's label and selection. The clear-button is displayed next to drop down button, and can be hidden, always visible or visible only if there is a selection or get the focus. The [Background\(exClearButtonUp\)](#) property specifies the visual appearance for the clear button, when it is up, while the [Background\(exClearButtonDown\)](#) property specifies the visual appearance for the clear button, when it is down. The [DropDownButtonWidth](#) property retrieves or sets the width, in pixels, to display the drop down button of the control (including the clear-button).



(the x indicates the clear-button)

The ShowClearButton property supports the following values:

- -1, the clear-button is always shown
- 0, the clear-button is not visible
- 1, the clear-button is visible only if the control's label is not empty or there are selected-items
- 2, the clear-button is visible only if the control has the focus
- 3 (1 OR 2), the clear-button is visible only if the control's label is not empty or there are selected-items (1) and the control has the focus (2)

For instance, the ShowClearButton property on 3 (1 OR 2) specifies that the clear-button is shown only if required (the control's label is not empty) and the control has the focus.

property ComboBox.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 control draws a thin rectangle around the focused item.

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 [SelectedItem](#) property to get the selected item. Use the [SelectItem](#) to select or unselect a specified item.

property ComboBox.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 [Replacelcon](#) 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 [RadiolImage](#) property to specify a different look for checkboxes or radio buttons in the cells.



property ComboBox.ShowLockedItems as Boolean

Retrieves or sets a value that indicates whether the locked/fixed items are visible or hidden.

*/*not supported in the lite version*/*

Type	Description
Boolean	A boolean expression that specifies whether the locked items are shown or hidden.

A locked or fixed item is always displayed on the top or bottom side of the control no matter if the control's list is scrolled up or down. Use the ShowLockedItems property to show or hide the locked items. Use the [LockedItemCount](#) property to add or remove items fixed/locked to the top or bottom side of the control. Use the [LockedItem](#) property to access a locked item by its position. Use the [CellCaption](#) property to specify the caption for a cell. Use the [LockedItemCount](#) property to lock or unlock items to the top or bottom side of the control's list.

method `ComboBox.ShowToolTip` (`ToolTip as String`, [`Title as Variant`], [`Alignment as Variant`], [`X as Variant`], [`Y as Variant`])

Shows the specified tooltip at given position.

Type	Description
ToolTip as String	<p>The ToolTip parameter can be any of the following:</p> <ul style="list-style-type: none">• <code>NULL(BSTR)</code> or "<code><null></code>"(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	<p>The Title parameter can be any of the following:</p> <ul style="list-style-type: none">• missing (<code>VT_EMPTY</code>, <code>VT_ERROR</code> type) or "<code><null></code>" (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	<p>A long expression that indicates the alignment of the tooltip relative to the position of the cursor. If missing (<code>VT_EMPTY</code>, <code>VT_ERROR</code>) the alignment of the tooltip for the object being hovered is not changed.</p> <p>The Alignment parameter can be one of the following:</p> <ul style="list-style-type: none">• 0 - <code>exTopLeft</code>• 1 - <code>exTopRight</code>• 2 - <code>exBottomLeft</code>• 3 - <code>exBottomRight</code>• 0x10 - <code>exCenter</code>• 0x11 - <code>exCenterLeft</code>• 0x12 - <code>exCenterRight</code>• 0x13 - <code>exCenterTop</code>• 0x14 - <code>exCenterBottom</code> <p>By default, the tooltip is aligned relative to the top-left corner (0 - <code>exTopLeft</code>).</p>

Specifies the horizontal position to display the tooltip as one of the following:

- missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current horizontal position of the cursor (current x-position)
- a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position)
- a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)

X as Variant

Specifies the vertical position to display the tooltip as one of the following:

- missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current vertical position of the cursor (current y-position)
- a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position)
- a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Y as Variant

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the [MouseMove](#) event. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

For instance:

- [ShowToolTip\(<null>, <null>, , +8, +8\)](#), shows the tooltip of the object moved relative

to its default position

- `ShowToolTip(<null>, 'new title')`, adds, changes or replaces the title of the object's tooltip
- `ShowToolTip('new content')`, adds, changes or replaces the object's tooltip
- `ShowToolTip('new content', 'new title')`, shows the tooltip and title at current position
- `ShowToolTip('new content', 'new title', '+8', '+8')`, shows the tooltip and title moved relative to the current position
- `ShowToolTip('new content', '', 128, 128)`, displays the tooltip at a fixed position
- `ShowToolTip('', '')`, hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

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

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

- `exp`, stores the plain text to be shown once the user clicks the anchor, such as "`<a ;exp=show lines>`"
- `e64`, encodes in BASE64 the HTML text to be shown once the user clicks the anchor, such as "`<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY `" 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=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **number[:width]** 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.
- **key[:width]** 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 **&**; (&), **<**; (<), **>**; (>), **"**; (") and **&#number;**; (the character with specified code), For instance, the **€** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <off 6>subscript" displays the text such as: Text with subscript The "Text with <off -6>superscript" displays the text such as: Text with subscript

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

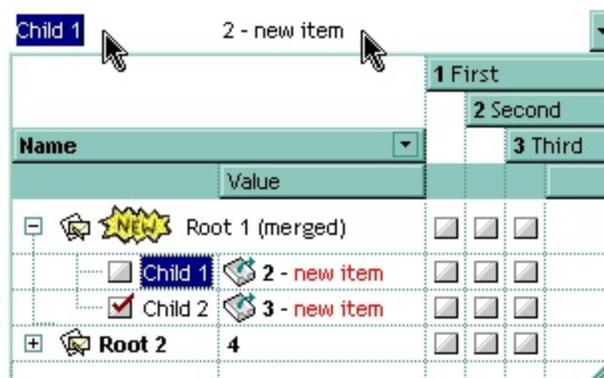
property ComboBox.SingleEdit as Boolean

Retrieves or sets a value that indicates whether the control displays a single edit control or multiple edit controls.

Type	Description
Boolean	A boolean expression that indicates whether the control displays one or more edit controls.

The number of the edit controls created by the control is equivalent with the number of Column objects. When SingleEdit property is True, then the control contains only a single edit control. Using TAB and Shift + TAB keys the user can navigate through the control columns ([UseTabKey](#) property). Use the [SearchColumnIndex](#) property specifies the index of the column where the incremental searching feature is on. At one moment only one edit control can have the focus. The focused edit control corresponds to the column that has its index equal with the SearchColumnIndex property. Use the [AdjustSearchColumn](#) property to allow displaying a hidden column in the control's label area (this is useful to allow owner draw in the control's label area) . Use the [LabelHeight](#) property to specify the height of the control's label area. Use the [Value](#) property to get or set the selected value, when the control contains a single column. Use the [Select](#) property to get or set the selected value when control contains multiple columns. The [LabelColumnIndex](#) property specifies a different column (index) to be displayed on the control's label, while the SingleEdit property is True.

The following screen shot shows the control when SingleEdit property is **False**. As you can see the control's label displays "Child 1" and "2 - new item" that indicates the values on the selected item:



The following screen shot shows the control when SingleEdit property is **True**. As you can see the control's label displays only the "Child 1" value:

Child 1		1 First		
		2 Second		
Name	Value	3 Third		
<input type="checkbox"/>  NEW Root 1 (merged)			<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Child 1	 2 - new item		<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Child 2	 3 - new item		<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>  Root 2	4		<input type="checkbox"/>	<input type="checkbox"/>

property ComboBox.SingleSel as Boolean

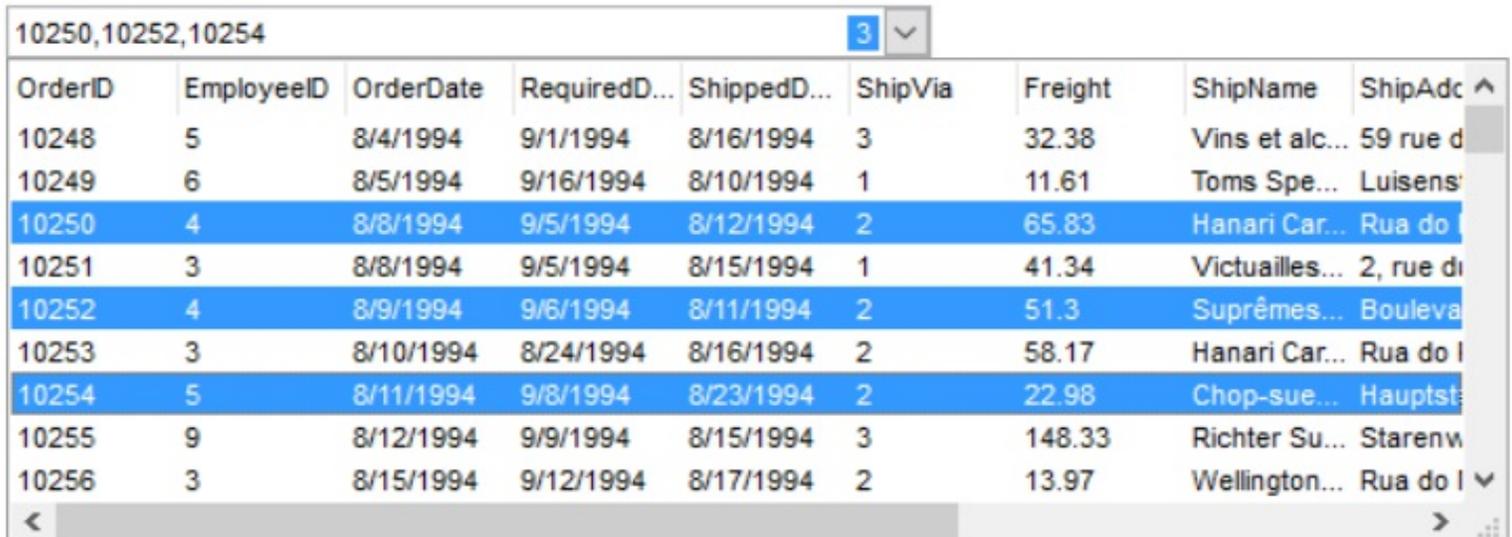
Retrieves or sets a value that indicates whether the control supports single or multiple selection.

Type	Description
Boolean	A Boolean expression that specifies whether the control supports single or multiple-selection.

By default, the SingleSel property is True, which indicates that the selection supports a single item only. Use the SingleSel property to allow the user to select multiple values/items/rows. The [SelectCount](#) property counts the selected items in the control. Use the [SelectedItem](#) property to retrieve the handle of the selected item. Use the [Value](#) property to select a value in a single column control. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectItem](#) or [Select](#) property to select an item. Use the [SelectAll](#), [UnselectAll](#) method to select / unselect all items within the control (mostly when the SingleSel property is False).

The control's label displays the list of selected values (separated by comma) if:

- SingleSel property is False
- [Style](#) property is DropDownList
- [SingleEdit](#) property is True
- [LabelColumnIndex](#) or [SearchColumnIndex](#) property points to a valid column



The screenshot shows a data grid with a label at the top left displaying '10250,10252,10254'. To the right of the label is a dropdown menu showing the number '3'. The data grid below has the following columns: OrderID, EmployeeID, OrderDate, RequiredD..., ShippedD..., ShipVia, Freight, ShipName, and ShipAdc. The rows are numbered 10248 to 10256. The rows for OrderID 10250, 10252, and 10254 are highlighted in blue.

OrderID	EmployeeID	OrderDate	RequiredD...	ShippedD...	ShipVia	Freight	ShipName	ShipAdc
10248	5	8/4/1994	9/1/1994	8/16/1994	3	32.38	Vins et alc...	59 rue d
10249	6	8/5/1994	9/16/1994	8/10/1994	1	11.61	Toms Spe...	Luisens
10250	4	8/8/1994	9/5/1994	8/12/1994	2	65.83	Hanari Car...	Rua do
10251	3	8/8/1994	9/5/1994	8/15/1994	1	41.34	Victuailles...	2, rue di
10252	4	8/9/1994	9/6/1994	8/11/1994	2	51.3	Suprêmes...	Bouleva
10253	3	8/10/1994	8/24/1994	8/16/1994	2	58.17	Hanari Car...	Rua do l
10254	5	8/11/1994	9/8/1994	8/23/1994	2	22.98	Chop-sue...	Hauptst
10255	9	8/12/1994	9/9/1994	8/15/1994	3	148.33	Richter Su...	Starenw
10256	3	8/15/1994	9/12/1994	8/17/1994	2	13.97	Wellington...	Rua do l

The [LabelText](#) property specifies the HTML format to display the count of selected items (shown on the right side of the label **3**), when control supports multiple-selection.

property ComboBox.SingleSort as Boolean

Returns or sets a value that indicates whether the control supports sorting by single or multiple columns. */*not supported in the lite version*/*

Type	Description
Boolean	A boolean expression that indicates whether the control supports sorting by single or multiple columns.

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 ComboBox.SortBarCaption as String

Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns. */*not supported in the lite version*/*

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**
- **<i> ... </i>** 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=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **number[:width]** 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.

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

Drag a column header here to sort by that column.	
	1 First
	2 Second
	3 Th...
Name ▼	Val... ▼

property ComboBox.SortBarColumnWidth as Long

Specifies the maximum width a column can be in the control's sort bar. */*not supported in the lite version*/*

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 ComboBox.SortBarHeight as Long

Retrieves or sets a value that indicates the height of the control's sort bar. */*not supported in the lite version*/*

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 ComboBox.SortBarVisible as Boolean

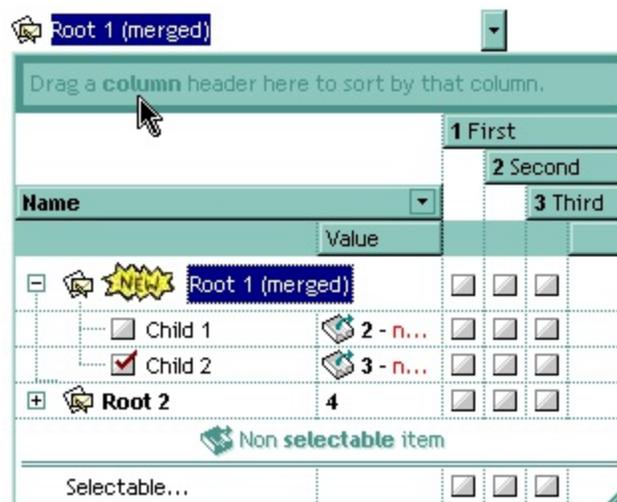
Retrieves or sets a value that indicates whether control's sort bar is visible or hidden. */*not supported in the lite version*/*

Type	Description
Boolean	A boolean expression that indicates whether the sort bar is visible or hidden.

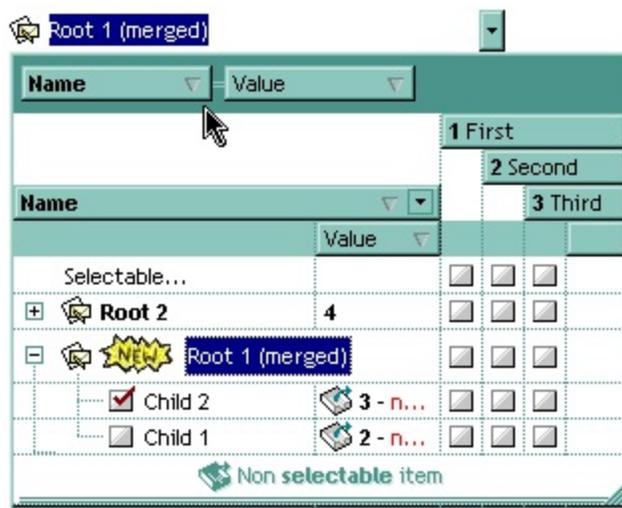
Use the SortBarVisible property to show the control's sort bar. By default, the SortBarVisible property is False. Use the [SingleSort](#) property to specify whether the control supports sorting by single or multiple columns. Sorting by a single column in the control is a simple matter of clicking on 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:

- 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.

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 ComboBox.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 whether the user clicks the column's header.

Use the SortOnClick property to specify the action that the control performs when the user clicks the column's header. Use the [SortChildren](#) method to sort a column, at runtime. Use the [DisplaySortIcon](#) property to hide the sort icon if the column is sorted. Use the [DefaultSortOrder](#) property to specify the column's default sort order, when the user first clicks the column's header. The control fires the [ColumnClick](#) event when the user clicks the column's header.

There are two methods to get the items sorted like follows:

- Using the [SortOrder](#) property of the [Column](#) object::

```
ComboBox1.Columns(ColIndex).SortOrder = SortAscending
```

The SortOrder property adds the sorting icon to the column's header, if the [DisplaySortIcon](#) property is True.

- Using the [SortChildren](#) method of the [Items](#) collection. The SortChildren sorts the items. The SortChildren method sorts the child items of the given parent item in the control. SortChildren will not recourse through the tree, only the immediate children of the item will be sorted. The following sample sorts descending the list of root items on the "Column 1"(if your control displays a list, all items are considered being root items).

```
ComboBox1.Items.SortChildren 0, "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 ComboBox.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 61 = 50,752 (63.78%)
Control: 1 x 14,856 = 14,856 (18.67%)
Column: 13 x 664 = 8,632 (10.85%)
Item: 64 x 72 = 4,608 (5.79%)
Items: 1 x 624 = 624 (0.78%)
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 101 = 84,032 (63.74%)
Control: 1 x 24,296 = 24,296 (18.43%)
Column: 13 x 1,072 = 13,936 (10.57%)
Item: 64 x 128 = 8,192 (6.21%)
Items: 1 x 1,192 = 1,192 (0.90%)
Columns: 1 x 136 = 136 (0.10%)
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 ComboBox.Style as StyleEnum

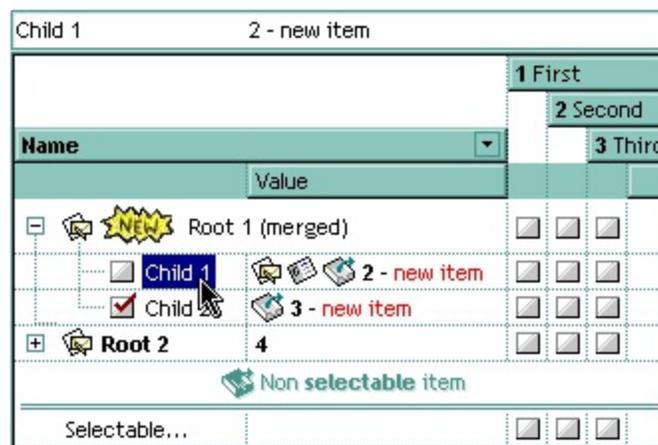
Retrieves or sets a value that indicates the control's behavior.

Type	Description
StyleEnum	A StyleEnum expression that indicates the control's style.

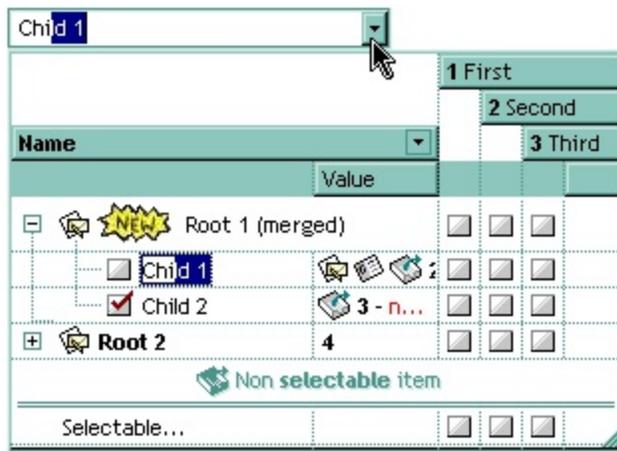
The control supports three styles: Simple, DropDown and DropDownList. When control's style is Simple, the list is always displayed. When the control's style is DropDown or DropDownList the list is visible only if the user presses the F4 key, or clicks the drop down button of the control. Use the [Key](#) property assign a new key to open the drop down window when user presses a key. Use the [Columns](#) property to access the control's columns collection. Use the [Items](#) property to access the control's items collection. Use the [DropDown](#) method to show programmatically the drop down portion of the control. Use the [AutoDropDown](#) property to specify whether the user shows the drop down portion of the control. Use the [Add](#) method to add new columns to the control. Use the [AddItem](#), [InsertItem](#), [PutItems](#), [DataSource](#) properties to add new items to the control. Use the [BeginUpdate](#) and [EndUpdate](#) methods to maintain performance while adding new columns and items. Use the [Background](#) property to change the drop down button's visual appearance.

Note! The Style property can be changed at runtime, starting with the version 1.0.5.7. If you are creating and removing the control dynamically and you need to set the appropriate Style property at runtime make sure that you are loading data **after** setting the Style property, else the data is lost. The Style property clears the columns and items collections.

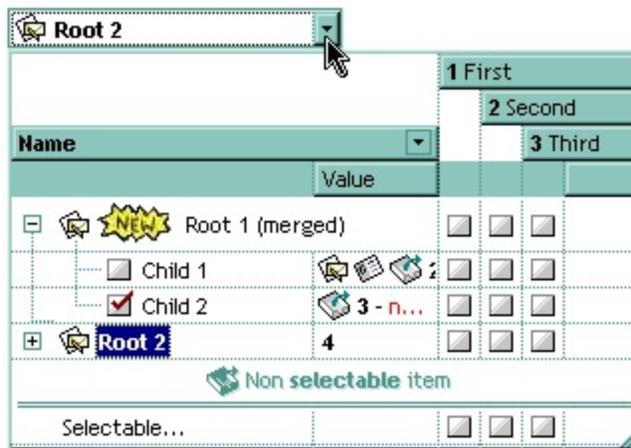
The following screen shot shows the control when Style property is **Simple** (The control doesn't provide a drop down button):



The following screen shot shows the control when Style property is **DropDown** (The control provides a drop down button, and user may use the edit controls in the control's label area to type values that are not in the drop down list.):



The following screen shot shows the control when Style property is **DropDownList** (The control provides a drop down button, and user can select only values that are in the drop down portion of the control.):



property ComboBox.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). The [ExecuteTemplate](#) property gets the result of executing a template script.

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

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

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

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

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable = property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values*

separated by commas. (Sample: `h = InsertItem(0, "New Child")`)

- *property(list of arguments) = value Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- *method(list of arguments) Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- *{ Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- *} Ending the object's context*
- *object. property(list of arguments).property(list of arguments).... The .(dot) character splits the object from its property. For instance, the `Columns.Add("Column1").HeaderBackColor = RGB(255,0,0)`, adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: `BackColor = RGB(255,0,0)`*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier*

property ComboBox.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 below:

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 ComboBox.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
NewVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / [TemplateDef](#) property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

The [TemplateDef](#), TemplatePut, [Template](#) and [ExecuteTemplate](#) support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable = property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property(list of arguments) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method(list of arguments) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- { *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- } *Ending the object's context*
- object.property(list of arguments).property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the*

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may use constant expressions as follows:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may start with 0x which indicates a hexa decimal representation, else it should start with a digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by # character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by " or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also, the template or x-script code may support general functions as follows:

- **Me** property indicates the original object.
- **RGB(R,G,B)** property retrieves an RGB value, where the R, G, B are byte values that indicate the R G B values for the color being specified. For instance, the following code changes the control's background color to red: *BackColor = RGB(255,0,0)*
- **LoadPicture(file)** property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.
- **CreateObject(progID)** property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.

property ComboBox.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. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. Use the [CellToolTip](#) property to specify the cell's tooltip. Use the [ShowToolTip](#) method to display a custom tooltip.

property ComboBox.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. 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 [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 [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 ComboBox.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.

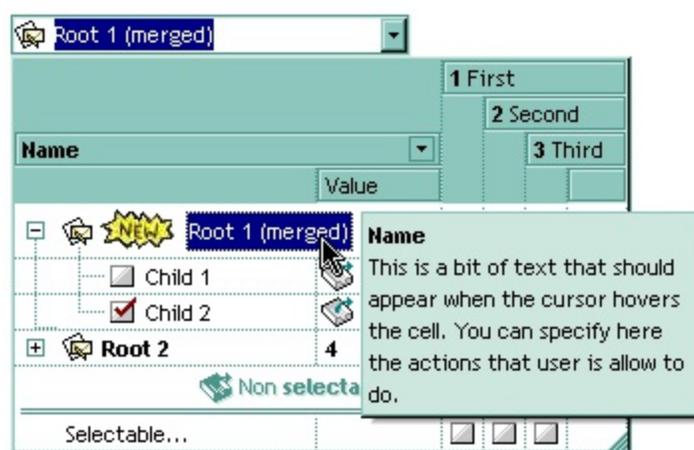
If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. The [ToolTipFont](#) property changes the 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 ComboBox.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, in pixels.

Use the ToolTipWidth property to specify the width of the tooltip window. The height of the tooltip window is computed based on the 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. The [ToolTipFont](#) property changes the font for the control's tooltip. Use the [CellToolTip](#) property to specify the cell's tooltip. Use the [ShowToolTip](#) method to display a custom tooltip.



property ComboBox.TreeColumnIndex as Long

Retrieves or sets a value indicating the column's index where the hierarchy will be displayed.

Type	Description
Long	A long expression that indicates the index of column where the hierarchy is displayed.

Use TreeColumnIndex property to change the column where the hierarchy lines are painted. Use [SearchColumnIndex](#) property to specify the column where the incremental search feature works. The control paints no hierarchy, if the TreeColumnIndex property is -1. Use the [HasLines](#), [LinesAtRoot](#) properties to show the hierarchy lines. The [Alignment](#), [CellHAlignment](#) property aligns cells in the column. If the index of the column is equal with the TreeColumnIndex property, the cells can be aligned only to left or to the right. Use the [Indent](#) property to define the amount, in pixels, that child items are indented relative to their parent items.

property ComboBox.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.

The following VB sample shows how to activate the control's unbound mode:

```
Option Explicit
Dim its As Items
Implements IUnboundHandler

' The ExComboBox invokes ItemsCount property, when the UnboundHandler property is
called.
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
    IUnboundHandler_ItemsCount = 50000
End Property

' The ExComboBox invokes ReadItem method when the Item ( ItemHandle, Index ) is
required.
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object,
ByVal ItemHandle As Long)
    its.CellCaption(ItemHandle, 0) = Index
    its.CellImage(ItemHandle, 0) = Index Mod 3 + 1
    its.CellHasCheckBox(ItemHandle, 0) = True
    ' Each 8th item has a different font attribute
    If (Index Mod 8) = 7 Then
        its.CellStrikeOut(ItemHandle, 0) = True
    End If
End Sub
```

```
Private Sub Form_Load()
    With ComboBox1
```

```
.BeginUpdate
```

```
' Initializes few control properties
```

```
.ColumnAutoResize = True
```

```
.SingleEdit = True
```

```
.MarkSearchColumn = False
```

```
.HeaderVisible = False
```

```
' Holds the Items object into a variable, to avoid calling
```

```
' Items property every time when ReadItem handler is invoked.
```

```
' Internally, the Items property invokes a QueryInterface method that can slow app's speed.
```

```
Set its = .Items
```

```
.Columns.Add "1"
```

```
' Sets the control's unbound handler. The Form implements the IUnboundHandler interface
```

```
Set .UnboundHandler = Me
```

```
.EndUpdate
```

```
End With
```

```
End Sub
```

The following VC sample shows how to activate the control's unbound mode. The sample requires a combo control into your dialog, and needs a new member variable associated to the control. (in this case the MFC class wizard will generate for you a class wrapper).

```
m_exCombo.BeginUpdate();  
m_exCombo.SetColumnAutoResize( TRUE );  
m_exCombo.SetSingleEdit( TRUE );  
m_exCombo.SetHeaderVisible( FALSE );  
m_exCombo.GetColumns().Add( "1" );  
m_exCombo.SetUnboundHandler( &m_unboundHandler.m_xHandler );  
m_exCombo.EndUpdate();
```

The m_unboundHandler is an object of CUnboundHandler type.

Here's the definition for CUnboundHandler class:

```
/******
```

REVISION LOG ENTRY

Revision By: Exontrol

Revised on 4/14/2002 12:44:39 PM

Comments: Implements IUnboundHandler interface

This code may be used in compiled form in any way you desire. This file may not be redistributed modified or unmodified without the authors written consent.

```
*****/
```

```
#if !defined(_UNBOUNDHANDLER_)
```

```
#define _UNBOUNDHANDLER_
```

```
// The IUnboundHandler interface is not included by class wizard so, we need to import it  
from the control's type library
```

```
#import "c:\winnt\system32\excombobox.dll" rename( "GetItems", "exGetItems" )
```

```
class CUnboundHandler : public CCmdTarget
```

```
{
```

```
public:
```

```
    CUnboundHandler();
```

```
    virtual ~CUnboundHandler();
```

```
    DECLARE_INTERFACE_MAP()
```

```
public:
```

```
    BEGIN_INTERFACE_PART(Handler, EXCOMBOBOXLib::IUnboundHandler)
```

```
        STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
```

```
        STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source, long ItemHandle);
```

```
    END_INTERFACE_PART(Handler)
```

```
#ifdef _USESMARTSOURCE
```

```
    EXCOMBOBOXLib::IComboBoxPtr m_spComboBox;
```

```
    EXCOMBOBOXLib::IItemsPtr m_spltems;
```

```
#endif // _USESMARTSOURCE
```

```
};
```

```
#endif // !defined(_UNBOUNDHANDLER_)
```

Here's the implementation for CUnboundHandler class:

```
/*  
*****  
REVISION LOG ENTRY  
Revision By: Exontrol  
Revised on 4/14/2002 12:44:39 PM  
Comments: Implements IUnboundHandler interface  
This code may be used in compiled form in any way you desire. This  
file may not be redistributed modified or unmodified without the authors  
written consent.  
*****  
*/
```

```
#include "stdafx.h"
```

```
#include "UnboundHandler.h"
```

```
////////////////////////////////////////////////////////////////  
// class CUnboundHandler  
////////////////////////////////////////////////////////////////
```

```
BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
```

```
    INTERFACE_PART(CUnboundHandler, __uuidof(EXCOMBOBOXLib::IUnboundHandler),  
Handler)
```

```
END_INTERFACE_MAP()
```

```
// Function name : CUnboundHandler::CUnboundHandler
```

```
// Description : Default constrcutor
```

```
// Return type :
```

```
CUnboundHandler::CUnboundHandler()
```

```
{
```

```
}
```

```
// Function name : CUnboundHandler::~CUnboundHandler
```

```
// Description : Virtual destructor
```

```
// Return type :
```

```
CUnboundHandler::~CUnboundHandler()
```

```
{  
}
```

```
// Function name : CUnboundHandler::Handler::get_ItemsCount
```

```
// Description : Gets the items count
```

```
// Return type : STDMETHODIMP
```

```
// Argument : long* pVal
```

```
STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch * Source, long*
```

```
pVal)
```

```
{
```

```
    METHOD_PROLOGUE(CUnboundHandler, Handler);
```

```
    if ( pVal )
```

```
    {
```

```
        *pVal = 150000;
```

```
        return S_OK;;
```

```
    }
```

```
    return E_POINTER;
```

```
}
```

```
// Function name : CUnboundHandler::XHandler::raw_ReadItem
```

```
// Description : Reads the item
```

```
// Return type : STDMETHODIMP
```

```
// Argument : long Index
```

```
// Argument : IDispatch * Source
```

```
// Argument : long ItemHandle
```

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index, IDispatch *
```

```
Source, long ItemHandle)
```

```
{
```

```
    METHOD_PROLOGUE(CUnboundHandler, Handler);
```

```
#ifndef _USESMARTSOURCE
```

```
    // The Source points to a ComboBox object.
```

```
    EXCOMBOBOXLib::IItemsPtr spltems = NULL;
```

```

EXCOMBOBOXLib::IComboBoxPtr spCombo = NULL;
if ( SUCCEEDED( Source->QueryInterface( &spCombo ) ) )
    if ( SUCCEEDED( spCombo->get_Items( &spltems ) ) )
    {
        _variant_t vtHandle(ItemHandle), vtColumn( (long)0 ) ;
        TCHAR szCaption[1024] = _T("");
        wsprintf( szCaption, _T("%i"), Index );
        spltems->CellCaption[vtHandle][vtColumn] = _variant_t(szCaption);
    }
#else
    // To avoid calling QueryInterface and get_Items every time when an item is reading,
    // you can hold two smart pointers in the parent class ( CUnboundHandler ) like this:
    //
    // public: EXCOMBOBOXLib::IComboBoxPtr m_spComboBox;
    // public: EXCOMBOBOXLib::IItemsPtr m_spltems;
    //
    // You need to initialize the m_spComboBox, and m_spltems before calling
SetUnboundHandler, like this:
    // m_unboundHandler.m_spComboBox = m_exCombo.GetControlUnknown();
    // m_unboundHandler.m_spltems = m_exCombo.GetItems().m_lpDispatch;
    // m_exCombo.SetUnboundHandler( &m_unboundHandler.m_xHandler );
    _variant_t vtHandle(ItemHandle), vtColumn( (long)0 ) ;
    TCHAR szCaption[1024] = _T("");
    wsprintf( szCaption, _T("%i"), Index );
    pThis->m_spltems->CellCaption[vtHandle][vtColumn] = _variant_t(szCaption);
#endif
    return S_OK;
}

```

```

// Function name : CUnboundHandler::XHandler::QueryInterface
// Description : Implements the IUnknown::QueryInterface method
// Return type : STDMETHODCALLTYPE
// Argument : REFIID riid
// Argument : void** ppvObject

```

```

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( EXCOMBOBOXLib::IUnboundHandler), riid ))
    {
        *ppvObject = static_cast<EXCOMBOBOXLib::IUnboundHandler*>( this );
        AddRef();
        return S_OK;
    }
    return E_NOINTERFACE;
}
return E_POINTER;
}

```

// Function name : CUnboundHandler::XHandler::AddRef

// Description : Implements the IUnknown::AddRef property

// Return type : STDMETHODCALLTYPE

// Argument : REFIID riid

// Argument : void** ppvObject

STDMETHODCALLTYPE CUnboundHandler::XHandler::AddRef()

```

{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}

```

// Function name : CUnboundHandler::XHandler::QueryInterface

// Description : Implements IUnknown:: Release property

// Return type :

// Argument : ULONG

STDMETHODCALLTYPE CUnboundHandler::XHandler::Release()

```

{

```

```
METHOD_PROLOGUE(CUnboundHandler, Handler);
```

```
return 0;
```

```
}
```

property ComboBox.UseMnemonic as Boolean

Gets or sets a value indicating whether the control interprets an ampersand character (&) in the item's caption to be an access key prefix character.

Type	Description
Boolean	A boolean expression indicating whether the control interprets an ampersand character (&) in the item's caption to be an access key prefix character.

By default, the UseMnemonic property is True. Use the UseMnemonic property to let control displays & characters.

property ComboBox.UseTabKey as Boolean

Retrieves or sets a value that indicates whether the control uses the TAB key for changing the searching column.

Type	Description
Boolean	A boolean expression that indicates whether the control uses the TAB key for changing the searching column.

By default, the UseTabKey property is True. By default, the TAB key is used to change the searching column. The [SearchColumnIndex](#) property to specify the index of the searching column. The control provides incremental search feature, so user can type characters and the control looks for items that match the typed characters. Use the [AutoSearch](#) property to enable or disable the incremental searching. Use the [MarkSearchColumn](#) property to hide the rectangle around the searching column. Use the UseTabKey property to allow to container navigating through its controls. Use the [AdjustSearchColumn](#) property to allow displaying a hidden column in the control's label area. Use the [Visible](#) property to hide a column.

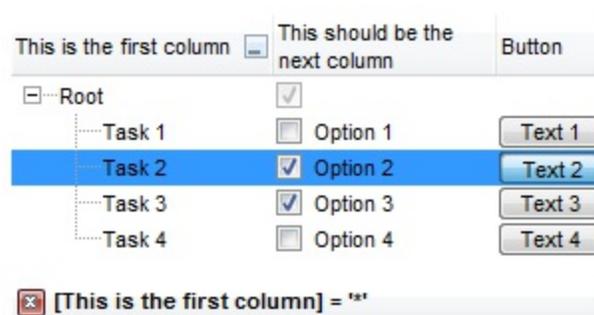
property ComboBox.UseVisualStyle as UIVisualThemeEnum

Specifies whether the control uses the current visual theme to display certain UI parts.

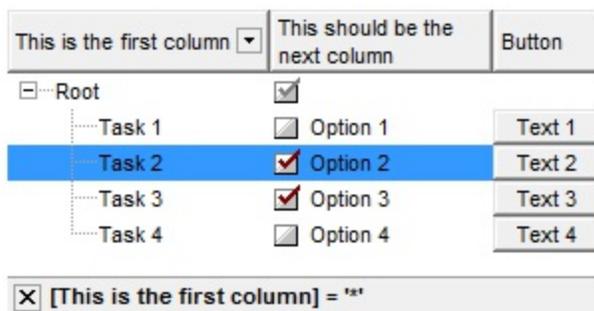
Type	Description
UIVisualStyleEnum	An UIVisualThemeEnum expression that specifies which UI parts of the control are shown using the current visual theme.

By default, the UseVisualStyle property is exDefaultVisualStyle, which means that all known UI parts are shown as in the current theme. The UseVisualStyle 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 UseVisualStyle property has effect only a current theme is selected for your desktop. The UseVisualStyle 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 UseVisualStyle property is exDefaultVisualStyle:



since the second screen shot shows the same data as the UseVisualStyle property is exNoVisualStyle:



property ComboBox.Value as Variant

Specifies the selected value for controls with a single column.

Type	Description
Variant	A Variant expression that indicates the selected value.

Use the Value property to get or set the selected value, when the control contains a single column. Use the [Select](#) property to get or set the selected value when control contains multiple columns. If the control contains multiple columns, the Value property get or set the selected value on the column pointed by [SearchColumnIndex](#) property. Use the [SelectItem](#) method to select an item given its handle. Use the [ItemByIndex](#) property to access an item given its index. The [SelectionChanged](#) event is fired when user changes the control's selection. Use the [EditText](#) property to retrieve the text inside the control's label.

The following VB samples are equivalents:

```
With ComboBox1
    .Select(.SearchColumnIndex) = "Child 2"
End With
```

and

```
With ComboBox1
    .Value = "Child 2"
End With
```

The following VB sample selects the "Root" item:

```
With ComboBox1
    .BeginUpdate
        .HeaderVisible = False
        .ColumnAutoResize = True
        .Columns.Add "Column 1"
        With .Items
            .InsertItem .AddItem("Root"), , "Child"
        End With
        .Value = "Root"
    .EndUpdate
End With
```

The following C++ sample selects the "Child 2" item:

```
m_combobox.SetValue( COleVariant( "Child 2" ) );
```

The following VB.NET sample selects the "Child 2" item:

```
With AxComboBox1  
    .Value = "Child 2"  
End With
```

The following C# sample selects the "Child 2" item:

```
axComboBox1.Value = "Child 2";
```

The following VFP sample selects the "Child 2" item:

```
with thisform.ComboBox1  
    .Object.Value = "Child 2"  
endwith
```

property `ComboBox.Version` as `String`

Retrieves the control's version.

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

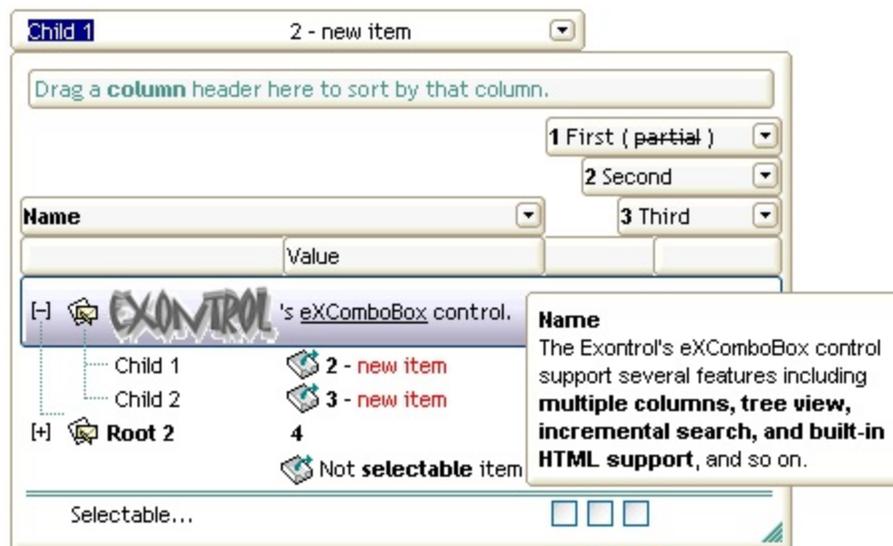
The `Version` property specifies the control's version.

property ComboBox.VisualAppearance as Appearance

Retrieves the control's appearance. */*not supported in the lite version*/*

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 **borders** using the [Appearance](#) property
- visual effect of the borders of the drop down portion of the control, using the [DropDownBorder](#) property
- borders for the control's **tooltip**, using the [Background\(exToolTipAppearance\)](#) property
- control's **header bar**, [HeaderBackColor](#) 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
- **"drop down" button**, cell's **button**, **"drop down" filter bar button**, **"close" filter bar button**, **scrollbars**, and so on, [Background](#) property
- [CellImage](#), [CellImages](#), [HeaderImage](#), [CheckImage](#) or [RadiolImage](#), [HasButtonsCustom](#) property

property `ComboBox.WidthList`([Reserved as Variant]) as Long

Specifies the width for the drop-down window.

Type	Description
Reserved as Variant	Reserved.
Long	A long expression that indicates the width of the drop down window.

Use the `WidthList` property to specify the width of the drop down window. The `WidthList` property has no effect if the control's [Style](#) is Simple. Use the [HeightList](#) property to specify the height of the drop down window. The [MinWidthList](#) property specifies a value that indicates the minimum width of the control's drop down window. Use the [AllowSizeGrip](#) property to specify whether the drop down portion of the control is resizable at runtime. Use the [ColumnAutoResize](#) property to specify whether the visible columns should fit the control's client area.

The following VB sample specifies the width of the drop down portion of the control:

```
ComboBox1.WidthList() = 100
```

The following C++ sample specifies the width of the drop down portion of the control:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;  
m_combobox.SetWidthList( vtMissing, 100 );
```

The following VB.NET sample specifies the width of the drop down portion of the control:

```
AxComboBox1.set_WidthList(100)
```

The following C# sample specifies the width of the drop down portion of the control:

```
axComboBox1.set_WidthList(100);
```

The following VFP sample the width of the drop down portion of the control:

```
with thisform.ComboBox1.Object  
    .WidthList("") = 100  
endwith
```

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 ComboBox1.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_combobox.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 AxComboBox1.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):

```
EXCOMBOBOXLib.ConditionalFormat cf =  
axComboBox1.ConditionalFormats.Add("%1+%2<%0",null);  
cf.Bold = true;  
cf.ApplyTo = (EXCOMBOBOXLib.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.ComboBox1.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 ComboBox1.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_combobox.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 AxComboBox1.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):

```
EXCOMBOBOXLib.ConditionalFormat cf =
axComboBox1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
```

```
cf.ApplyTo = (EXCOMBOBOXLib.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.ComboBox1.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. A constant is a float expression (for instance, 23.45). 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 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. When 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 [CellCaption](#) 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 fourth 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 fourth 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 strikethrough.
- [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:

```
ComboBox1.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_combobox.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:

```
AxComboBox1.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:

```
axComboBox1.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.ComboBox1.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 ComboBox1.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 ComboBox1.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_combobox.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 AxComboBox1.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):

```
EXCOMBOBOXLib.ConditionalFormat cf =
axComboBox1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Italic = true;
```

```
cf.ApplyTo = (EXCOMBOBOXLib.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.ComboBox1.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 ComboBox1.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_combobox.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 AxComboBox1.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):

```
EXCOMBOBOXLib.ConditionalFormat cf =  
axComboBox1.ConditionalFormats.Add("%1+%2<%0",null);  
cf.Bold = true;  
cf.ApplyTo = (EXCOMBOBOXLib.FormatApplyToEnum)1;
```

The following VFP sample applies strikethrough 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.ComboBox1.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 ComboBox1.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_combobox.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 AxComboBox1.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):

```
EXCOMBOBOXLib.ConditionalFormat cf =  
axComboBox1.ConditionalFormats.Add("%1+%2<%0",null);  
cf.Underline = true;
```

```
cf.ApplyTo = (EXCOMBOBOXLib.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.ComboBox1.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 strikethrough.
- [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:

```
ComboBox1.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 ComboBox1.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_combobox.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_combobox.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:

```
AxComboBox1.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 AxComboBox1.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:

```
axComboBox1.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):

```
EXCOMBOBOXLib.ConditionalFormat cf =  
axComboBox1.ConditionalFormats.Add("%1+%2<%0",null);  
cf.Bold = true;  
cf.ApplyTo = (EXCOMBOBOXLib.FormatApplyToEnum)1;
```

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

```
thisform.ComboBox1.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.ComboBox1.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 ComboBox1.ConditionalFormats
    Debug.Print c.Expression
Next
```

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

```
Dim i As Integer
With ComboBox1.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_combobox.GetConditionalFormats().GetCount(); i++ )
{
    CConditionalFormat cf = m_combobox.GetConditionalFormats().GetItem( COleVariant( i
));
    OutputDebugString( cf.GetExpression() );
}
```

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

```
Dim c As EXCOMBOBOXLib.ConditionalFormat
For Each c In AxComboBox1.ConditionalFormats
    System.Diagnostics.Debug.Write(c.Expression)
```

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

```
Dim i As Integer
With AxComboBox1.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 (EXCOMBOBOXLib.ConditionalFormat c in axComboBox1.ConditionalFormats)
    System.Diagnostics.Debug.Write(c.Expression);
```

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

```
for (int i = 0; i < axComboBox1.ConditionalFormats.Count; i++)
    System.Diagnostics.Debug.Write(axComboBox1.ConditionalFormats[i].Expression);
```

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

```
with thisform.ComboBox1.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 ComboBox1.ConditionalFormats
    Debug.Print c.Expression
Next
```

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

```
Dim i As Integer
With ComboBox1.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_combobox.GetConditionalFormats().GetCount(); i++ )
{
    CConditionalFormat cf = m_combobox.GetConditionalFormats().GetItem( COleVariant( i
));
    OutputDebugString( cf.GetExpression() );
}
```

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

```
Dim c As EXCOMBOBOXLib.ConditionalFormat
For Each c In AxComboBox1.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 AxComboBox1.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 (EXCOMBOBOXLib.ConditionalFormat c in axComboBox1.ConditionalFormats)
    System.Diagnostics.Debug.Write(c.Expression);
```

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

```
for (int i = 0; i < axComboBox1.ConditionalFormats.Count; i++)
    System.Diagnostics.Debug.Write(axComboBox1.ConditionalFormats[i].Expression);
```

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

```
with thisform.ComboBox1.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.

Items object

The Items object contains a collection of items. Each item is identified by a handle HITEM. The HITEM is of long type. Each item contains a collection of cells. The number of cells is determined by the number of Column objects into 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 collection can be organized as a hierarchy or as a simple list. The Items object supports the following properties and methods:

Name	Description
AcceptSetParent	Retrieves a value indicating whether the SetParent method can be accomplished.
AddItem	Adds a new item, and returns a handle to the newly created item.
CellBackColor	Retrieves or sets the cell's background color.
CellBold	Retrieves or sets a value that specifies whether the cell's caption should appear in bold.
CellCaption	Retrieves or sets the text displayed on a specific cell.
CellCaptionFormat	Specifies how the cell's caption is displayed.
CellChecked	Retrieves the cell's handle that is checked on a specific radio group.
CellData	Retrieves or sets a value that indicates an 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.
CellImage	Retrieves or sets a value that indicates the index of cell's image into Images collection.

CellImages	Specifies an additional list of icons shown in the cell.
CellItalic	Retrieves or sets a value that specifies whether the cell's caption should appear in italic.
CellItem	Retrieves the handle of item that is the owner of a specific cell.
CellMerge	Retrieves or sets a value that indicates the index of the cell that's merged to.
CellOwnerDraw	Specifies the cell's owner draw handler.
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 radio or check type.
CellStrikeOut	Retrieves or sets a value that specifies 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 specifies whether the cell's caption is underlined.
CellVAlignment	Retrieves or sets a value that indicates how the cell's caption is vertically aligned.
ChildCount	Retrieves the number of children items.
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.
DefaultItem	Retrieves or sets the default item's handle.
	Returns or sets a value that determines whether a item

[EnableItem](#) can respond to user-generated events.

[EnsureVisibleItem](#) Ensures the given item is in the visible client area.

[ExpandItem](#) Expands, or collapses, the child items of the specified item.

[FindItem](#) Finds an item, looking for Caption in ColIndex column. The searching starts at StartIndex item.

[FindPath](#) Finds the item, given its path. The control searches the path on the SearchColumnIndex column.

[FirstVisibleItem](#) Retrieves the handle of the first visible item into control.

[FocusItem](#) Retrieves the handle of item that has the focus.

[FormatCell](#) Specifies the custom format to display the cell's content.

[FullPath](#) Returns the fully qualified path of the referenced item in the control. The caption is taken from the column SearchColumnIndex.

[InsertItem](#) Inserts a new item, and returns a handle to the newly created item.

[IsItemLocked](#) Returns a value that indicates whether the item is locked or unlocked.

[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 specifies whether the item should appear in bold.

[ItemByIndex](#) Retrieves the handle of the item given its index in Items collection..

[ItemCell](#) Retrieves a value that indicates the cell's handle based on a specific column.

[ItemChild](#) Retrieves the child of a specified item.

[ItemCount](#) Retrieves the number of items.

[ItemData](#) Retrieves or sets a value that indicates an extra data for a specific item.

[ItemDivider](#) Specifies whether the item acts like a divider item. The value indicates the index of column used to define the divider's title.

ItemDividerLine	Defines the type of line in the divider item.
ItemDividerLineAlignment	Specifies the alignment of the line in the divider item.
ItemFont	Retrieves or sets the item's font.
ItemForeColor	Retrieves or sets a foreground color for a specific item.
ItemHasChildren	Adds an expand button to left side of the item even if the item has no child items.
ItemHeight	Retrieves or sets the item's height.
ItemItalic	Retrieves or sets a value that specifies 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.
ItemParent	Returns the handle of the item's parent item.
ItemPosition	Retrieves or sets a value that indicates the item's position in the children list.
ItemStrikeOut	Retrieves or sets a value that specifies whether the item should appear in strikeout.
ItemToIndex	Retrieves the index of item into Items collection given its handle.
ItemUnderline	Retrieves or sets a value that specifies whether the item is underlined.
LastVisibleItem	Retrieves the handle of the last visible item.
LockedItem	Retrieves the handle of the locked item.
LockedItemCount	Specifies the number of items fixed on the top or bottom side of the control.
MatchItemCount	Retrieves the number of items that match the filter.
MergeCells	Merges a list of cells.
NextSiblingItem	Retrieves the next sibling of the item in the parent's child list.
NextVisibleItem	Retrieves the handle of next visible item.
PathSeparator	Returns or sets the delimiter character used for the path returned by the FullPath property.
PrevSiblingItem	Retrieves the previous sibling of the item in the parent's child list.

PrevVisibleItem	Retrieves the handle of previous visible item.
RemoveAllItems	Removes all items from the control.
RemoveItem	Removes a specific item.
RemoveSelection	Removes the selected items (including the descendents).
RootCount	Retrieves the number of root objects into Items collection.
RootItem	Retrieves the handle of the root item giving its index into the root items collection.
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 handle given its index into the selected items collection.
SelectItem	Selects or unselects a specific item.
SelectPos	Selects items by position.
SetParent	Changes the parent of the given item.
SortableItem	Specifies whether the item is sortable.
SortChildren	Sorts the child items of the given parent item in the control. SortChildren will not recurse through the tree, only the immediate children of Item will be sorted.
UnmergeCells	Unmerges a list of cells.
UnselectAll	Unselects all items.
VisibleCount	Retrieves the number of items being visible in the control's client area.
VisibleItemCount	Retrieves the number of visible items.

property Items.AcceptSetParent (Item as HITEM, NewParent as HITEM) as Boolean

Verifies whether the item can be the child of another item

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
NewParent as HITEM	A long expression that indicates the handle of the parent item.
Boolean	A boolean expression that indicates whether the Item can be child of the NewParent item.

The AcceptSetParent property doesn't change the parent item. Use the [SetParent](#) method to change the item's parent. Use the [ItemParent](#) property to retrieve the item's parent. Use the [InsertItem](#) method to add child items to another item. An item is called root, if it has no parent (ItemParent() gets 0).

method Items.AddItem ([Caption as Variant])

Adds a new item, and returns a handle to the newly created item.

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

Return	Description
HITEM	A long value that indicates the handle of the newly created item.

Use the AddItem method to add new items to the control. The AddItem method fails, if the control contains no columns. Use the [Add](#) method to add new columns to the control. The control fires [InsertItem](#) event when a new items is added to the control's list. The Caption parameter indicates the caption of the newly inserted item for the first column. If you have more than a column, you need to use the [CellCaption](#) property to assign a caption for each column. The Caption supports built-in HTML format if the [CellCaptionFormat](#) property is exHTML. Use [InsertItem](#) method to insert a child item. The InsertItem(,[Caption]) and AddItem([Caption]) are equivalents. Use the [PutItems](#) method to load an array of elements. Use the [LockedItemCount](#) property to lock or unlock items to the top or bottom side of the control's list. Use the [MergeCells](#) method to combine two or more cells in a single cell. Use the [SelectableItem](#) property to specify whether the user can select the item. 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 exComputedField type specifies a computed cell.

The following VB sample loads items from a safe array:

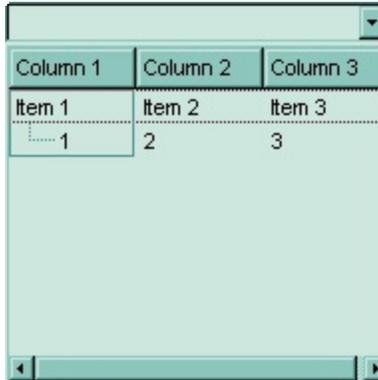
```
With ComboBox1
    .BeginUpdate

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

With .Items
    Dim h As HITEM
    h = .AddItem(Array("Item 1", "Item 2", "Item 3"))
```

```
.InsertItem h, , Array(1, 2, 3)
.ExpandItem(h) = True
End With
```

```
.EndUpdate
End With
```



Column 1	Column 2	Column 3
Item 1	Item 2	Item 3
1	2	3

The following VB sample adds items to a single column control:

```
With ComboBox1
.BeginUpdate
.ColumnAutoSize = True
.HeaderVisible = False
.LinesAtRoot = exLinesAtRoot
.HasButtons = exCircle
.Columns.Add "Column 1"
With .Items
Dim h As HITEM
h = .AddItem("Item 1")
.InsertItem h, , "Item 1.1"
.InsertItem h, , "Item 1.2"
.ExpandItem(h) = True
End With
.EndUpdate
End With
```

The following C++ sample adds new items to the control:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
```

```
long iNewItem = items.AddItem( COleVariant( "Item 1" ) );
items.SetCellCaption( COleVariant( iNewItem ), COleVariant( (long)1 ), COleVariant(
"SubItem 1" ) );
iNewItem = items.AddItem( COleVariant( "Item 2" ) );
items.SetCellCaption( COleVariant( iNewItem ), COleVariant( (long)1 ), COleVariant(
"SubItem 2" ) );
```

The following VB.NET sample adds new items to the control:

```
With AxComboBox1.Items
    Dim iNewItem As Integer = .AddItem("Item 1")
    .CellCaption(iNewItem, 1) = "SubItem 1"
    iNewItem = .AddItem("Item 2")
    .CellCaption(iNewItem, 1) = "SubItem 2"
End With
```

The following C# sample adds new items to the control:

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;
int iNewItem = items.AddItem( "Item 1" );
items.set_CellCaption( iNewItem, 1, "SubItem 1" );
iNewItem = items.AddItem( "Item 2" );
items.set_CellCaption( iNewItem, 1, "SubItem 2" );
```

The following VFP sample adds new items to the control:

```
with thisform.ComboBox1.Items
    .DefaultItem = .AddItem("Item 1")
    .CellCaption(0, 1) = "SubItem 1"
    .DefaultItem = .AddItem("Item 2")
    .CellCaption(0, 1) = "SubItem 2"
endwith
```

property Items.CellBackColor([Item as Variant], [ColIndex as Variant]) as Color

Retrieves or sets the cell's background color.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the 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.

To change the background color for the entire item you can use [ItemBackColor](#) property. Use the [ClearCellBackColor](#) method to clear the cell's background color. Use the [BackColor](#) property to specify the control's background color. Use the [CellForeColor](#) property to specify the cell's foreground color. Use the [ItemForeColor](#) property to specify the item's foreground color. Use the [Def\(exCellBackColor\)](#) property to specify the background color for all cells in the 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.

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

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 first cell:

```
axComboBox1.Items.set_CellBackColor(axComboBox1.Items.FirstVisibleItem,
axComboBox1.SearchColumnIndex, ToUInt32(Color.Red));
```

The following VB.NET sample changes the background color for the first cell:

```
With AxComboBox1.Items
    .CellBackColor(.FirstVisibleItem, AxComboBox1.SearchColumnIndex) =
    ToUInt32(Color.Red)
End With
```

The following C++ sample changes the background color for the first cell:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
items.SetCellBackColor( COleVariant( items.GetFirstVisibleItem() ), COleVariant(
m_combobox.GetSearchColumnIndex() ), RGB(255,0,0) );
```

The following VB sample changes the background color for the first cell:

```
With ComboBox1.Items
    .CellBackColor(.FirstVisibleItem, ComboBox1.SearchColumnIndex) = vbRed
End With
```

The following VFP sample changes the background color for the first cell:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FirstVisibleItem
    .CellBackColor( 0, thisform.ComboBox1.SearchColumnIndex ) = RGB(255,0,0)
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
```

```
.Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
.Items.CellBold(.Items(0), 0) = True
```

```
.Items.CellBold(.Items(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
.Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

```
End With
```

property Items.CellBold([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value that specifies whether the cell should appear in bold.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell should appear in bold.

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 [CellCaptionFormat](#) 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
ComboBox1.BeginUpdate
With ComboBox1.Items
For Each h In ComboBox1.Items
    .CellBold(h, 0) = True
Next
End With
ComboBox1.EndUpdate
```

The following C++ sample bolds the focused cell:

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

The following C# sample bolds the focused cell:

```
axComboBox1.Items.set_CellBold(axComboBox1.Items.FocusItem, 0, true);
```

The following VB.NET sample bolds the focused cell:

```
With AxComboBox1.Items  
    .CellBold(.FocusItem, 0) = True  
End With
```

The following VFP sample bolds the focused cell:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellBold( 0, 0 ) = .t.  
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1  
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True  
    .Items.CellBold(.Items(0), 0) = True  
    .Items.CellBold(.Items(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True  
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True  
End With
```

property Items.CellCaption([Item as Variant], [ColIndex as Variant]) as Variant

Retrieves or sets the text displayed on a specific cell.

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

The CellCaption property specifies the cell's caption. To associate an user data for a cell you can use [CellData](#) property. The [CellCaptionFormat](#) property specifies whether the cell's caption displays a simple text, uses built-in HTML tags, or indicates a computed field. Use the [ItemData](#) property to associate an extra data to an item. Use the [Visible](#) property of the [Column](#) object to hide a column. The [AddItem](#) method specifies also the caption for the first cell in the item. Use the `` HTML tag to insert icons inside the cell's caption, if the [CellCaptionFormat](#) property is exHTML. Use the [CellImage](#) or [CellImages](#) property to assign an icon to a cell. Use the [CellPicture](#) property to assign a custom size picture to a cell. 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 exComputedField type specifies a computed cell.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
```

```
.Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
.Items.CellBold(.Items(0), 0) = True
```

```
.Items.CellBold(.Items(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
.Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

```
End With
```


property Items.CellCaptionFormat([Item as Variant], [ColIndex as Variant]) as CaptionFormatEnum

Specifies how the cell's caption is displayed.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index or cell's handle, or a string expression that specifies the column's key or the column's caption.
CaptionFormatEnum	A CaptionFormatEnum expression that defines the way how the cell's caption is displayed.

The component supports built-in HTML format. That means that you can use HTML tags when displays the cell's caption . By default, the CellCaptionFormat property is exText. If the CellCaptionFormat is exText, the cell displays the [CellCaption](#) property like it is. If the CellCaptionFormat is exHTML, the cell displays the CellCaption property using the HTML tags specified in the CaptionFormatEnum type. Use the [Def\(exCellCaptionFormat\)](#) 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.

property Items.CellChecked (RadioGroup as Long) as HCELL

Retrieves the handle of the cell that is checked, given the radio group identifier.

Type	Description
RadioGroup as Long	A long expression that indicates the radio group identifier.
HCELL	A long expression that indicates the cell's handle. Use the CellItem property to retrieve the handle of the owner item.

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 all cells on the first column into a radio group, and display the cell's checked on the radio group when the state of a radio group is changed:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellHasRadioButton(Item, 0) = True
        .CellRadioGroup(Item, 0) = 1234
    End With
End Sub

Private Sub ComboBox1_CellStateChanged(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    With ComboBox1.Items
        Debug.Print "In the 1234 radio group the """" & .CellCaption(, .CellChecked(1234)) &
"""" is checked."
    End With
End Sub
```

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_combobox.GetItems();
m_combobox.BeginUpdate();
```

```

for ( long i = 0; i < items.GetItemCount(); i++ )
{
    COleVariant vtItem( items.GetItemByIndex( i ) );
    items.SetCellHasRadioButton( vtItem, vtColumn, TRUE );
    items.SetCellRadioGroup( vtItem, vtColumn, 1234 );
}
m_combobox.EndUpdate();

```

```

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 OnCellStateChangedCombobox1(long Cell)
{
    if ( ::IsWindow( m_combobox.m_hWnd ) )
    {
        CItems items = m_combobox.GetItems();
        long hCell = items.GetCellChecked( 1234 );
        COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
        OutputDebugString( V2S( &items.GetCellCaption( vtMissing, COleVariant( hCell ) ) ) );
    }
}

```

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

```

With AxComboBox1
    .BeginUpdate()

```

```
With .Items
```

```
Dim k As Integer
```

```
For k = 0 To .ItemCount - 1
```

```
    .CellHasRadioButton(.ItemByIndex(k), 0) = True
```

```
    .CellRadioGroup(.ItemByIndex(k), 0) = 1234
```

```
Next
```

```
End With
```

```
.EndUpdate()
```

```
End With
```

```
Private Sub AxComboBox1_CellStateChanged(ByVal sender As Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent) Handles  
AxComboBox1.CellStateChanged, AxComboBox2.CellStateChanged
```

```
    With AxComboBox1.Items
```

```
        Debug.WriteLine(.CellCaption(, .CellChecked(1234)))
```

```
    End With
```

```
End Sub
```

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

```
axComboBox1.BeginUpdate();
```

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;
```

```
for (int i = 0; i < items.ItemCount; i++)
```

```
{
```

```
    items.set_CellHasRadioButton(items[i], 0, true);
```

```
    items.set_CellRadioGroup(items[i], 0, 1234);
```

```
}
```

```
axComboBox1.EndUpdate();
```

```
private void axComboBox1_CellStateChanged(object sender,
```

```
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent e)
```

```
{
```

```
    string strOutput = axComboBox1.Items.get_CellCaption(0,
```

```
axComboBox1.Items.get_CellChecked(1234)).ToString();
```

```
    strOutput += " state = " + axComboBox1.Items.get_CellState(null, e.cell).ToString();
```

```
    System.Diagnostics.Debug.WriteLine(strOutput);
```

```
}
```

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

```
thisform.ComboBox1.BeginUpdate()
with thisform.ComboBox1.Items
  local i
  for i = 0 to .ItemCount - 1
    .DefaultItem = .ItemByIndex(i)
    .CellHasRadioButton( 0,0 ) = .t.
    .CellRadioGroup(0,0) = 1234
  next
endwith
thisform.ComboBox1.EndUpdate()
```

Note : The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
  .Items.CellBold( .Items.ItemCell(.Items(0), 0)) = True
  .Items.CellBold(.Items(0), 0) = True
  .Items.CellBold(.Items(0)) = True
  .Items.CellBold(.Items.ItemByIndex(0)) = True
  .Items.CellBold(.Items.ItemByIndex(0), 0) = True
  .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
End With
```

property Items.CellData([Item as Variant], [ColIndex as Variant]) as Variant

Specifies the cell's extra data.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A variant expression that indicates the cell's user 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.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

With ComboBox1

```
.Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
.Items.CellBold(.Items(0), 0) = True
```

```
.Items.CellBold(.Items(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
.Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

End With

property Items.CellEnabled([Item as Variant], [ColIndex as Variant]) as Boolean

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell is enabled or disabled.

Use the CellEnabled property to disable a cell. A disabled cell looks grayed. Use the [EnableItem](#) property to disable an item. Once that one cell is disabled it cannot be checked or clicked. Use the [SelectableItem](#) property to specify the user can select an item. To disable a column you can use [Enabled](#) property of the Column object.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
```

```
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
    .Items.CellBold(.Items(0), 0) = True
```

```
    .Items.CellBold(.Items(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

```
End With
```

property Items.CellFont ([Item as Variant], [ColIndex as Variant]) as IFontDisp

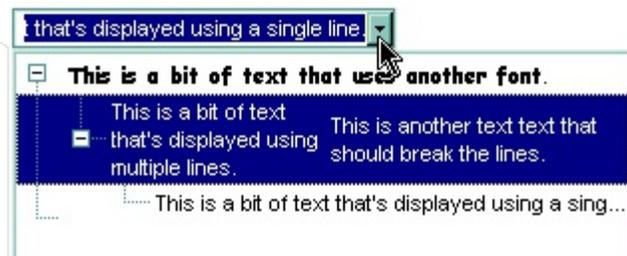
Retrieves or sets the cell's font.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or the column's key.
IFontDisp	A Font object that indicates the item's font.

By default, the CellFont property is nothing. If the CellFont property is nothing, 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 [CellCaptionFormat](#) to specify different font attributes. 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:

```
ComboBox1.BeginUpdate
With ComboBox1.Items
    .CellFont(.FocusItem, 0) = ComboBox1.Font
    With .CellFont(.FocusItem, 0)
        .Name = "Comic Sans MS"
        .Size = 10
        .Bold = True
    End With
End With
ComboBox1.EndUpdate
```



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

```
#include "Items.h"
#include "Font.h"
m_combobox.BeginUpdate();
CItems items = m_combobox.GetItems();
COleVariant vtItem(items.GetFocusItem()), vtColumn( (long)0 );
items.SetCellFont( vtItem, vtColumn, m_combobox.GetFont().m_lpDispatch );
```

```
COleFont font = items.GetCellFont( vtlItem, vtColumn );  
font.SetName( "Comic Sans MS" );  
font.SetBold( TRUE );  
m_combobox.EndUpdate();
```

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

```
AxComboBox1.BeginUpdate()  
With AxComboBox1.Items  
    .CellFont(.FocusItem, 0) = IFDH.GetIFontDisp(AxComboBox1.Font)  
    With .CellFont(.FocusItem, 0)  
        .Name = "Comic Sans MS"  
        .Bold = True  
    End With  
End With  
AxComboBox1.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:

```
axComboBox1.BeginUpdate();  
axComboBox1.Items.set_CellFont(axComboBox1.Items.FocusItem, 0,  
IFDH.GetIFontDisp(axComboBox1.Font));  
stdole.IFontDisp spFont =  
axComboBox1.Items.get_CellFont(axComboBox1.Items.FocusItem, 0);
```

```
spFont.Name = "Comic Sans MS";  
spFont.Bold = true;  
axComboBox1.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.ComboBox1.BeginUpdate()  
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellFont(0,0) = thisform.ComboBox1.Font  
with .CellFont(0,0)  
    .Name = "Comic Sans MS"  
    .Bold = .t.  
endwith  
endwith  
thisform.ComboBox1.EndUpdate()
```

property Items.CellForeColor([Item as Variant], [ColIndex as Variant]) as Color

Retrieves or sets the cell's foreground color.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Color	A color expression that indicates the cell's foreground color.

The CellForeColor property identifies the cell's foreground color. Use the [ClearCellForeColor](#) property to clear the cell's foreground color. Use the [ItemForeColor](#) property to specify the the item's foreground color. Use the [Def\(exCellForeColor\)](#) property to specify the foreground color for all cells in the 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.

For instance, the following VB code changes the left top cell of your control:
`ComboBox1.Items.CellForeColor(ComboBox1.Items(0), 0) = vbBlue`

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:

```
axComboBox1.Items.set_CellForeColor(axComboBox1.Items.FocusItem, 0,  
ToUInt32(Color.Red) );
```

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

```
With AxComboBox1.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_combobox.GetItems();  
items.SetCellForeColor( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),  
RGB(255,0,0) );
```

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

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellForeColor( 0, 0 ) = RGB(255,0,0)  
endwith
```

For instance, the following code shows how to change the left top cell of your control:
`ComboBox1.Items.CellForeColor(ComboBox1.Items(0), 0) = vbBlue`

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

With ComboBox1

.Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True

.Items.CellBold(.Items(0), 0) = True

.Items.CellBold(.Items(0)) = True

.Items.CellBold(.Items.ItemByIndex(0)) = True

.Items.CellBold(.Items.ItemByIndex(0), 0) = True

.Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True

End With

property Items.CellHorizontalAlignment ([Item as Variant], [ColIndex as Variant]) as AlignmentEnum

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

Type	Description
Item as Variant	A long expression that indicates the handle 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 CellHorizontalAlignment 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 [ClearCellHorizontalAlignment](#) method to clear the cell's alignment previously set by the CellHorizontalAlignment property. If the CellHorizontalAlignment property is not set, the Alignment property of the Column object indicates the cell's alignment. Please make sure that If the cell belongs to the column that displays the hierarchy in the control ([TreeColumnIndex](#) property indicates the index of the column that displays the hierarchy lines), the caption, icon or its picture can't be centered. In this case the caption can be aligned to the left or to the right. Use the [RightToLeft](#) property to align the drop down button to the left side of the control.

The following VB sample right aligns the focused cell:

```
With ComboBox1.Items
    .CellHorizontalAlignment(.FocusItem, 0) = AlignmentEnum.RightAlignment
End With
```

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

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

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

```
With AxComboBox1.Items
    .CellHorizontalAlignment(.FocusItem, 0) = EXCOMBOBOXLib.AlignmentEnum.RightAlignment
```

The following C# sample right aligns the focused cell:

```
axComboBox1.Items.set_CellHAlignment(axComboBox1.Items.FocusItem, 0,  
EXCOMBOBOXLib.AlignmentEnum.RightAlignment);
```

The following VFP sample right aligns the focused cell:

```
with thisform.ComboBox1.Items  
  .DefaultItem = .FocusItem  
  .CellHAlignment(0,0) = 2 && RightAlignment  
endwith
```

property Items.CellHasButton([Item as Variant], [ColIndex as Variant]) as Boolean

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell contains a 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 [CellCaption](#) property. Use the [Def](#) property to assign buttons for all cells in the column.

The following VB sample sets the cells of the first column to be of button type, and displays a message when one of them has been clicked.

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    ComboBox1.Items.CellHasButton(Item, 0) = True
End Sub

Private Sub ComboBox1_CellButtonClick(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    MsgBox "The user clicks the cell's button. " & ComboBox1.Items.CellCaption(, Cell)
End Sub
```

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

```
With ComboBox1.Items
    .CellHasButton(.FocusItem, 0) = True
End With
```

The following C++ sample assigns a button to the focused cell:

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

```
);
```

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

```
With AxComboBox1.Items  
    .CellHasButton(.FocusItem, 0) = True  
End With
```

The following C# sample assigns a button to the focused cell:

```
axComboBox1.Items.set_CellHasButton(axComboBox1.Items.FocusItem, 0, true);
```

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

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellHasButton(0,0) = .t.  
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1  
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True  
    .Items.CellBold(.Items(0), 0) = True  
    .Items.CellBold(.Items(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True  
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True  
End With
```

property Items.CellHasCheckBox([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value indicating whether the cell has an associated checkbox.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell contains a check box button.

To change the state for a check 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 radio type you have call [CellHasRadioButton](#) property. 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 [CheckImage](#) property to change the check box appearance. Use the [PartialCheck](#) property to allow partial check feature within the column. Use the [FilterType](#) property on exCheck to filter for checked or unchecked items.

The following sample enumerates the cells in the first column and assign a checkbox to all of them:

```
Dim h As Variant
ComboBox1.BeginUpdate
With ComboBox1.Items
For Each h In ComboBox1.Items
    .CellHasCheckBox(h, 0) = True
Next
End With
ComboBox1.EndUpdate
```

The same thing we can do using the Def property like follows:

```
With ComboBox1.Columns(0)
    .Def(exCellHasCheckBox) = True
End With
```

The following sample shows how to set the type of cells to radio type while adding new items:

```
Private Sub ComboBox1_AddItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    ComboBox1.Items.CellHasCheckBox(Item, 0) = True
End Sub
```

The following sample shows how to use the CellStateChanged event to display a message when a cell of radio or check type has changed its state:

```
Private Sub ComboBox1_CellStateChanged(ByVal Item As EXCOMBOBOXLibCtl.HITEM,
ByVal ColIndex As Long)
    Debug.Print "The cell """" & ComboBox1.Items.CellCaption(Item, ColIndex) & """" has
changed its state. The new state is " & If(ComboBox1.Items.CellState(Item, ColIndex) = 0,
"Unchecked", "Checked")
End Sub
```

The following VB sample adds a checkbox to the focused cell:

```
With ComboBox1.Items
    .CellHasCheckBox(.FocusItem, 0) = True
End With
```

The following C++ sample adds a checkbox to the focused cell:

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

The following C# sample adds a checkbox to the focused cell:

```
axComboBox1.Items.set_CellHasCheckBox(axComboBox1.Items.FocusItem, 0, true);
```

The following VB.NET sample adds a checkbox to the focused cell:

```
With AxComboBox1.Items
    .CellHasCheckBox(.FocusItem, 0) = True
End With
```

The following VFP sample adds a checkbox to the focused cell:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FocusItem
    .CellHasCheckBox(0,0) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
End With
```

property Items.CellHasRadioButton([Item as Variant], [ColIndex as Variant]) as Boolean

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell contains a 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 assigns a radio button to the focused cell:

```
With ComboBox1.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_combobox.GetItems();
items.SetCellHasRadioButton( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),
TRUE );
items.SetCellRadioGroup( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),
1234 );
```

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

```
With AxComboBox1.Items
```

```
.CellHasRadioButton(.FocusItem, 0) = True
```

```
.CellRadioGroup(.FocusItem, 0) = 1234
```

```
End With
```

The following C# sample assigns a radio button to the focused cell:

```
axComboBox1.Items.set_CellHasRadioButton(axComboBox1.Items.FocusItem, 0, true);
```

```
axComboBox1.Items.set_CellRadioGroup(axComboBox1.Items.FocusItem, 0, 1234);
```

The following VFP sample assigns a radio button to the focused cell:

```
with thisform.ComboBox1.Items
```

```
.DefaultItem = .FocusItem
```

```
.CellHasRadioButton(0,0) = .t.
```

```
.CellRadioGroup(0,0) = 1234
```

```
endwith
```

Call `ComboBox1.Items.CellCaption(, ComboBox1.Items.CellChecked(1234))` to get the cell's caption that's checked in the group 1234.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
```

```
.Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
.Items.CellBold(.Items(0), 0) = True
```

```
.Items.CellBold(.Items(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
.Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

```
End With
```

property Items.CellImage ([Item as Variant], [ColIndex as Variant]) as Long

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Long	A long value that indicates the index of the icon in Images collection. The Images collection is 1 based. 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 property to assign a single icon to the cell. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. Use the [CellImages](#) property to assign multiple icons to a cell. Use the [Images](#) method or [Replacelcon](#) property to update the list of icons in the control, at runtime. Use the CellImage() = 0 to remove an cell's icon. The [CellImageClick](#) event occurs when the cell's icon is clicked. The icon's size is always 16 x 16. Use the [CellPicture](#) property to assign a custom size picture 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 [ItemFromPoint](#) property to retrieve the part of the control being clicked. Use the [AssignEditImageOnSelect](#) property to display the cell's icon on the control's label when user selects a new item. Use the `` HTML tag to insert icons inside the cell's caption, if the [CellCaptionFormat](#) property is exHTML. Use the [FilterType](#) property on exImage to filter items by icons.

The following VB sample assigns the same icon to all cells in the first column:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellImage(Item, 0) = 1
    End With
End Sub
```

The following VB sample changes the cell's image when the user clicks on the cell's image (to run the following sample you have to add two images to the ComboBox's images collection),

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellImage(Item, 0) = 1
    End With
End Sub
```

```
Private Sub ComboBox1_CellImageClick(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    ComboBox1.Items.CellImage(, Cell) = ComboBox1.Items.CellImage(, Cell) Mod 2 + 1
End Sub
```

The following C++ sample assigns an icon to the focused cell:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
items.SetCellImage( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ), 1 );
```

The following VB.NET sample assigns an icon to the focused cell:

```
With AxComboBox1.Items
    .CellImage(.FocusItem, 0) = 1
End With
```

The following C# sample assigns an icon to the focused cell:

```
axComboBox1.Items.set_CellImage(axComboBox1.Items.FocusItem, 0, 1);
```

The following VFP sample assigns an icon to the focused cell:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FocusItem
    .CellImage(0,0) = 1
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a

cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
```

```
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
    .Items.CellBold(.Items(0), 0) = True
```

```
    .Items.CellBold(.Items(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

```
End With
```

property Items.CellImages ([Item as Variant], [ColIndex as Variant]) as Variant

Specifies an additional list of icons shown in the cell.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
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.
Variant	A string expression that indicates the list of icons shown in the cell.

Use the CellImages property to assign multiple icons to a cell. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. The [CellImage](#) property assign a single icon to the cell. 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 [Images](#) method or [Replacelcon](#) property to update the list of icons in the control, at runtime. The [CellImageClick](#) event occurs when the cell's image is clicked. Use the [ItemFromPoint](#) property to determine the index of the icon being clicked. Use the [CloseOnDbClick](#) property to specify whether the user closes the drop down portion of the control by a single or double click. Use the [CellPicture](#) property to load a custom size picture to a cell. Use the `` HTML tag to insert icons inside the cell's caption, if the [CellCaptionFormat](#) property is exHTML.

The following VB sample assign first and third icon to the cell:

```
With ComboBox1.Items
    .CellImages(.ItemByIndex(0), 1) = "1,3"
End With
```

The following VB sample displays the index of the icon being clicked in a cell:

```
Private Sub ComboBox1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim i As HITEM, h As HitTestInfoEnum, c As Long
    With ComboBox1
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
    End With
    If (i <> 0) Or (c <> 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 displays the index of the icon being clicked in a cell:

```

void OnMouseUpCombobox1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, i = m_combobox.GetItemFromPoint( X, Y, &c, &hit );
    if ( i != 0 )
        if ( 68 /*exHTCellIcon*/ == (hit & 68 /*exHTCellIcon*/ ) )
        {
            CString strOutput = V2S( &m_combobox.GetItems().GetCellCaption(COleVariant(i),
COleVariant(c) ) );
            strOutput += " Index = ";
            strOutput += V2S(COleVariant(hit >> 16));
            strOutput += "\r\n";
            OutputDebugString( strOutput );
        }
}

```

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

```

Private Sub AxComboBox1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib.IComboBoxEvents_MouseUpEvent) Handles
AxComboBox1.MouseUpEvent
    With AxComboBox1
        Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (i >= 0) Then
            If (hit And EXCOMBOBOXLib.HitTestInfoEnum.exHTCellIcon) =
EXCOMBOBOXLib.HitTestInfoEnum.exHTCellIcon Then
                Debug.WriteLine(.Items.CellCaption(i, c) & " Index = " & ((hit And &HFFFF0000)
/ 65536))
            End If
        End If
    End With
End Sub

```

```
End With
End Sub
```

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

```
private void axComboBox1_MouseUpEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_MouseUpEvent e)
{
    EXCOMBOBOXLib.HitTestInfoEnum hit;
    int c = 0, i = axComboBox1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i != 0)
        System.Diagnostics.Debug.WriteLine(axComboBox1.Items.get_CellCaption(i,
c).ToString() + " Index = " + (Convert.ToUInt32(hit) >> 16).ToString());
}
```

The following VFP sample displays the cell from the cursor:

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

local c, hit
c = 0
hit = 0
with thisform.ComboBox1
    .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
    if ( .Items.DefaultItem <> 0 )
        if ( bitand( hit, 68 )= 68 )
            wait window nowait .Items.CellCaption( 0, c ) + " Index=" + Str( Int((hit - 68)/65536) )
        endif
    endif
endwith
```

property Items.CellItalic([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value that specifies whether the cell should appear in italic.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell 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 [CellCaptionFormat](#) 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 ComboBox1.Items
    .CellItalic(.FocusItem, 0) = True
End With
```

The following C++ sample makes italic the focused cell:

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

The following C# sample makes italic the focused cell:

```
axComboBox1.Items.set_CellItalic(axComboBox1.Items.FocusItem, 0, true);
```

The following VB.NET sample makes italic the focused cell:

```
With AxComboBox1.Items
    .CellItalic(.FocusItem, 0) = True
End With
```

The following VFP sample makes italic the focused cell:

```
with thisform.ComboBox1.Items  
  .DefaultItem = .FocusItem  
  .CellItalic( 0, 0 ) = .t.  
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1  
  .Items.CellBold( .Items.ItemCell(.Items(0), 0) ) = True  
  .Items.CellBold(.Items(0), 0) = True  
  .Items.CellBold(.Items(0)) = True  
  .Items.CellBold(.Items.ItemByIndex(0)) = True  
  .Items.CellBold(.Items.ItemByIndex(0), 0) = True  
  .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True  
End With
```

property Items.CellItem (Cell as HCELL) as HITEM

Retrieves the handle of the item that is the owner of a specific cell.

Type	Description
Cell as HCELL	A long expression that indicates the cell's handle.
HITEM	A long expression that indicates the item's handle.

Use the `CellItem` property to retrieve the item's handle. Use the [ItemCell](#) property to get the cell's handle given an item and a column. Most of the properties of the `Items` object that have parameters [`Item as Variant`], [`ColIndex as Variant`], could use the handle of the cell to identify the cell, instead the `ColIndex` parameter. For instance the following statements are equivalents:

```
With ComboBox1.Items
    .CellCaption(.FocusItem, 0) = "this"
    .CellCaption(, .ItemCell(.FocusItem, 0)) = "this"
End With
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of `HCELL` type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

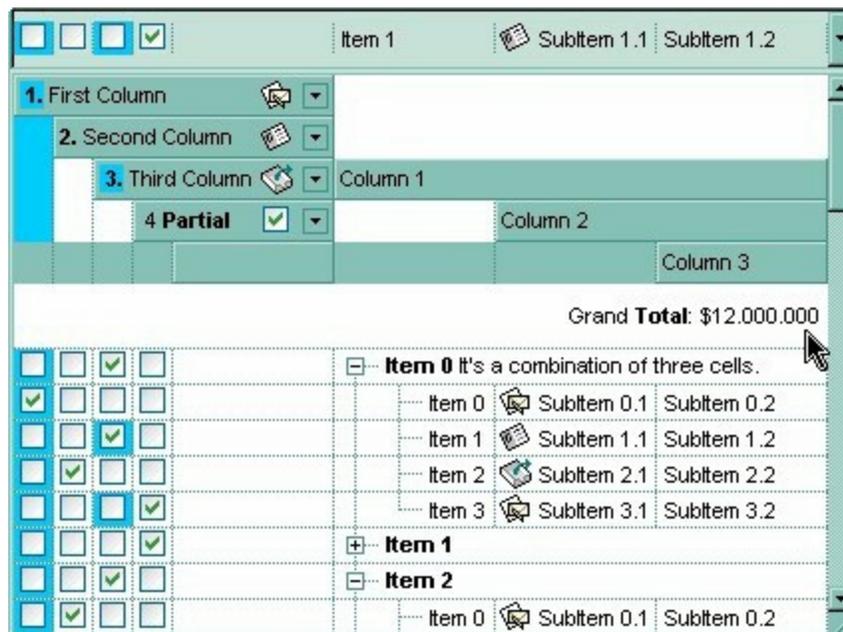
```
With ComboBox1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
End With
```

property Items.CellMerge([Item as Variant], [ColIndex as Variant]) as Variant

Retrieves or sets a value that indicates the index of the cell that's merged to. */*not supported in the lite version*/*

Type	Description
Item as Variant	A long expression that indicates the item's handle.
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.
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 [ItemDivider](#) property to display a single cell in the entire item. Use the [UnmergeCells](#) method to unmerge the merged cells. Use the CellMerge property to unmerge a single cell. Use the [MergeCells](#) method to combine one or more cells in a single cell. Use the [Add](#) method to add new columns to the control. Use the [LockedItemCount](#) property to add or remove items locked to the top or bottom side of the control.



The following sample shows few methods to unmerge cells:

```
With ComboBox1
```

```
With .Items
```

```
.UnmergeCells .ItemCell(.RootItem(0), 0)
```

```
End With
End With
```

```
With ComboBox1
  With .Items
    Dim r As Long
    r = .RootItem(0)
    .UnmergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1))
  End With
End With
```

```
With ComboBox1
  .BeginUpdate
  With .Items
    .CellMerge(.RootItem(0), 0) = -1
    .CellMerge(.RootItem(0), 1) = -1
    .CellMerge(.RootItem(0), 2) = -1
  End With
  .EndUpdate
End With
```

You can merge the first three cells in the root item using any of the following methods:

```
With ComboBox1
  With .Items
    .CellMerge(.RootItem(0), 0) = Array(1, 2)
  End With
End With
```

```
With ComboBox1
  .BeginUpdate
  With .Items
    Dim r As Long
    r = .RootItem(0)
    .CellMerge(r, 0) = 1
    .CellMerge(r, 0) = 2
  End With
  .EndUpdate
```

End With

With ComboBox1

.BeginUpdate

With .Items

Dim r As Long

r = .RootItem(0)

.MergeCells .ItemCell(r, 0), .ItemCell(r, 1)

.MergeCells .ItemCell(r, 0), .ItemCell(r, 2)

End With

.EndUpdate

End With

With ComboBox1

With .Items

Dim r As Long

r = .RootItem(0)

.MergeCells .ItemCell(r, 0), **Array**(.ItemCell(r, 1), .ItemCell(r, 2))

End With

End With

With ComboBox1

With .Items

Dim r As Long

r = .RootItem(0)

.MergeCells **Array**(.ItemCell(r, 0), .ItemCell(r, 1), .ItemCell(r, 2))

End With

End With

The following VB sample merges the first three cells in the focused item:

With ComboBox1.Items

.CellMerge(.FocusItem, 0) = 1

.CellMerge(.FocusItem, 0) = 2

End With

The following C++ sample merges the first three cells in the focused item:

```
#include "Items.h"  
CItems items = m_combobox.GetItems();  
COleVariant vtItem( items.GetFocusItem() ), vtColumn( long( 0 ) );  
items.SetCellMerge( vtItem, vtColumn, COleVariant( long(1) ) );  
items.SetCellMerge( vtItem, vtColumn, COleVariant( long(2) ) );
```

The following VB.NET sample merges the first three cells in the focused item:

```
With AxComboBox1.Items  
    .CellMerge(.FocusItem, 0) = 1  
    .CellMerge(.FocusItem, 0) = 2  
End With
```

The following C# sample merges the first three cells in the focused item:

```
axComboBox1.Items.set_CellMerge(axComboBox1.Items.FocusItem, 0, 1);  
axComboBox1.Items.set_CellMerge(axComboBox1.Items.FocusItem, 0, 2);
```

The following VFP sample merges the first three cells in the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellMerge(0,0) = 1  
    .CellMerge(0,0) = 2  
endwith
```

In other words, the sample shows how to display the first cell using the space occupied by three cells.

property Items.CellOwnerDraw ([Item as Variant], [ColIndex as Variant]) as IOwnerDrawHandler

Specifies the cell's owner draw handler.

Type	Description
Item as Variant	A long value that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index or the cell's handle (in case that the Item parameter is missing or if it is 0). A string expression that indicates the column's caption.
IOwnerDrawHandler	An object that implements the IOwnerDrawHandler interface.

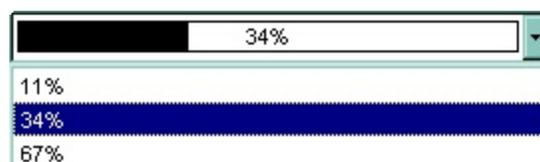
Use the CellOwnerDraw property to paint yourself the cell. The CellOwnerDraw property specifies whether the user is responsible for painting the cell. By default, the CellOwnerDraw property is nothing, and so the control does the painting. Using the notification interfaces is faster than using events. For instance, let's say that we need cells where for some values we need to get displayed other things. Use the [Def\(exCellOwneDraw\)](#) property to assign an owner draw object for the entire column. Use the [AdjustSearchColumn](#) property to specify whether a hidden column displays data in the control's label area.

How can I paint myself the control's label area? The idea is to let the control having an owner draw hidden column that's displayed in the control's label area.

You need the followings:

- [Style](#) property is **DropDownList** (required)
- [SingleEdit](#) property is True
- [ColumnAutoResize](#) property is True
- [AdjustSearchColumn](#) property is **False**
- A hidden column (use the [Visible](#) property to hide a column)
- [SearchColumnIndex](#) property should point to the hidden columns

So, a VB sample may look as follows:



Implements EXCOMBOBOXLibCtl.IOwnerDrawHandler

Private Type RECT

Left As Long

Top As Long

Right As Long

Bottom As Long

End Type

Private Const OPAQUE = 2

Private Const WHITE_BRUSH = 0

Private Const BLACK_BRUSH = 4

Private Const DT_CENTER = &H1;

Private Const DT_VCENTER = &H4;

Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long

Private Declare Function FrameRect Lib "user32" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long

Private Declare Function GetStockObject Lib "gdi32" (ByVal nIndex As Long) As Long

Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hdc As Long, ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As Long

Private Declare Function SetBkMode Lib "gdi32" (ByVal hdc As Long, ByVal nBkMode As Long) As Long

Private Sub Form_Load()

Dim iHidden As Long

With ComboBox1

.BeginUpdate

.BackColor = vbWhite

.BackColorEdit = .BackColor

.Style = DropDownList

.SingleEdit = True

.ColumnAutoResize = True

.AdjustSearchColumn = False

.MarkSearchColumn = False

.HeaderVisible = False

.Columns.Add "Column"

```

With .Columns.Add("")
    .Visible = False
    ComboBox1.SearchColumnIndex = .Index
End With
iHidden = .SearchColumnIndex
With .Items
    Set .CellOwnerDraw(.AddItem(Array("11%", 0.11)), iHidden) = Me
    Set .CellOwnerDraw(.AddItem(Array("34%", 0.34)), iHidden) = Me
    Set .CellOwnerDraw(.AddItem(Array("67%", 0.67)), iHidden) = Me
    .SelectItem(.ItemByIndex(0)) = True
End With
.EndUpdate
End With
End Sub

Private Sub IOwnerDrawHandler_DrawCell(ByVal hdc As Long, ByVal Left As Long, ByVal
Top As Long, ByVal Right As Long, ByVal Bottom As Long, ByVal Item As Long, ByVal
Column As Long, ByVal Source As Object)
    Dim r As RECT, rP As RECT, rT As RECT
    With r
        r.Left = Left + 2
        r.Right = Right - 2
        r.Top = Top
        r.Bottom = Bottom - 1
    End With
    rP = r
    rT = r
    Dim d As Double
    d = Source.Items.CellCaption(Item, Column)
    rP.Right = (rP.Right - rP.Left) * d + rP.Left

    FillRect hdc, rP, GetStockObject(BLACK_BRUSH)
    FrameRect hdc, r, GetStockObject(BLACK_BRUSH)

    Dim strText As String
    strText = Source.Items.CellCaption(Item, Column - 1)
    SetBkMode hdc, OPAQUE

```

```
rT.Top = rT.Top + 1
```

```
DrawText hdc, strText, Len(strText), rT, DT_VCENTER Or DT_CENTER
```

```
End Sub
```

property Items.CellPicture ([Item as Variant], [ColIndex as Variant]) as Variant

Retrieves or sets the cell's picture.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A Picture object being displayed on the cell (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 with a cell a check or radio button, an icon, a picture and a caption. Use the CellPicture property to associate a picture with a cell. You can use the CellPicture property when you want to display images with different widths in 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 specify the height of the item. The CellPictureWidth and CellPictureHeight properties specifies the size of the area where the cell's picture is stretched.

The following VB sample shows how to load a picture to a cell:

```
ComboBox1.Items.CellPicture(ComboBox1.Items.ItemByIndex(0), 0) =  
LoadPicture("c:/winnt/logo.gif")
```

or

```
ComboBox1.Items.CellPicture(ComboBox1.Items.ItemByIndex(0), 0) = "c:\winnt\logo.gif"
```

The following VB sample loads a picture to a cell using a BASE64 encoded strings:

```
With ComboBox1  
    .BeginUpdate  
  
    .ColumnAutoSize = True
```

```
.HeaderVisible = False
.Columns.Add "Column 1"
.BackColor = vbWhite
```

```
Dim h As HITEM
h = .Items.AddItem("Item 1")
```

```
Dim s As String
s =
```

```
"gBCJr+BAAg0HGwEgwog4jg4ig4BAEFg4AZEKisZjUbAAzg5mg6Zg7Mg7/g0ek8oGcgjsijsk
```

```
s = s +
```

```
"XgBadlDXdYSXRb9wWBclK2taF1gAl5HiPaN8oPdINWbaF23KAwyWkNYyXxg9p3WNYjU/c
```

```
.Items.CellPicture(h, 0) = s
.Items.ItemHeight(h) = 26
```

```
.EndUpdate
```

```
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;
                        _ASSERT( 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_combobox.GetItems();
    items.SetCellPicture( COleVariant( items.GetFocusItem() ), COleVariant(long(0)), vtPicture
);
}

```

```
pPicture->Release();  
}
```

The following VB.NET sample loads a picture from a file:

```
With AxComboBox1.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:

```
axComboBox1.Items.set_CellPicture(axComboBox1.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 GetPictureDisp(System.Drawing.Image image)
{
    return System.Windows.Forms.AxHost.GetPictureDispFromPicture( image );
}
}
```

The following VFP sample loads a picture from a file:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FocusItem
    .CellPicture( 0, 0 ) = LoadPicture("c:\winnt\zapotec.bmp")
endwith
```

property Items.CellPictureHeight ([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets a value that indicates the height of the cell's picture.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
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 ([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets a value that indicates the width of the cell's picture.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
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([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets a value indicating which radio group a cell is contained in.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Long	A long expression that identifies the cell's radio group.

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 shows how to set the radio type for all cells of the first column, and groups all of them in the same radio group (1234):

```
Dim h As Variant
ComboBox1.BeginUpdate
With ComboBox1.Items
For Each h In ComboBox1.Items
    .CellHasRadioButton(h, 0) = True
    .CellRadioGroup(h, 0) = 1234
Next
End With
ComboBox1.EndUpdate
```

or

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellHasRadioButton(Item, 0) = True
        .CellRadioGroup(Item, 0) = 1234
    End With
End Sub
```

To find out the radio cell that is checked in the radio group 1234 you can use: `MsgBox ComboBox1.Items.CellCaption(, ComboBox1.Items.CellChecked(1234))`

The following VB sample groups all cells of the first column into a radio group, and displays the checked cell:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellHasRadioButton(Item, 0) = True
        .CellRadioGroup(Item, 0) = 1234
    End With
End Sub
```

```
Private Sub ComboBox1_CellStateChanged(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    With ComboBox1.Items
        Debug.Print "In the 1234 radio group the """" & .CellCaption(, .CellChecked(1234)) &
"""" is checked."
    End With
End Sub
```

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

```
With ComboBox1.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_combobox.GetItems();
items.SetCellHasRadioButton( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),
TRUE );
items.SetCellRadioGroup( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),
1234 );
```

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

```
With AxComboBox1.Items
    .CellHasRadioButton(.FocusItem, 0) = True
```

```
.CellRadioGroup(.FocusItem, 0) = 1234
```

```
End With
```

The following C# sample assigns a radio button to the focused cell:

```
axComboBox1.Items.set_CellHasRadioButton(axComboBox1.Items.FocusItem, 0, true);  
axComboBox1.Items.set_CellRadioGroup(axComboBox1.Items.FocusItem, 0, 1234);
```

The following VFP sample assigns a radio button to the focused cell:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellHasRadioButton(0,0) = .t.  
    .CellRadioGroup(0,0) = 1234  
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

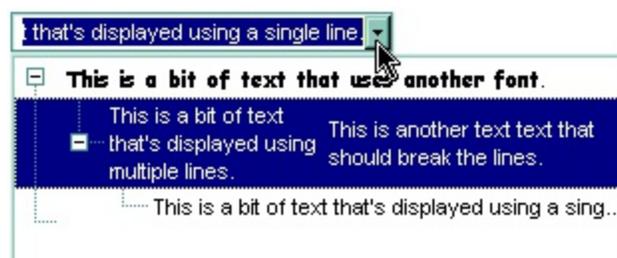
```
With ComboBox1  
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True  
    .Items.CellBold(.Items(0), 0) = True  
    .Items.CellBold(.Items(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True  
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True  
End With
```

property Items.CellSingleLine([Item as Variant], [ColIndex as Variant]) as CellSingleLineEnum

Retrieves or sets a value indicating whether the cell is painted using one line, or more than one line.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the 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. Use the [ItemAllowSizing](#) property to specify whether the user can resize the item at runtime.



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 ComboBox1.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_combobox.GetItems();  
items.SetCellSingleLine( COleVariant( items.GetFocusItem() ), COleVariant( long(0) ), FALSE  
);
```

The following VB.NET sample displays the caption of the focused cell using multiple lines:

```
With AxComboBox1.Items  
    .CellSingleLine(.FocusItem, 0) = False  
End With
```

The following C# sample displays the caption of the focused cell using multiple lines:

```
axComboBox1.Items.set_CellSingleLine(axComboBox1.Items.FocusItem, 0, false);
```

The following VFP sample displays the caption of the focused cell using multiple lines:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .CellSingleLine( 0, 0 ) = .f  
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1  
    .Items.CellBold( .Items.ItemCell(.Items(0), 0)) = True  
    .Items.CellBold(.Items(0), 0) = True  
    .Items.CellBold(.Items(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0)) = True  
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True  
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```


property Items.CellState([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets the cell's state. Affects only check and radio cells.

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

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 [PartialCheck](#) property to allow partial check feature within the column. 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 ComboBox1.Items
    .CellHasCheckBox(.FocusItem, 0) = True
    .CellState(.FocusItem, 0) = 1
End With
```

The following VB sample displays a message the user clicks a check box:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellHasCheckBox(Item, 0) = True
    End With
End Sub

Private Sub ComboBox1_CellStateChanged(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    With ComboBox1.Items
        Debug.Print "The cell """" & .CellCaption(, Cell) & """" has changed its state. The new
state is " & IIf(.CellState(, Cell) = 0, "Unchecked", "Checked")
    End With
End Sub
```

End Sub

The following C++ sample adds a check box that's checked to the focused cell:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
COleVariant vtItem( items.GetFocusItem() ), vtColumn( long(0) );
items.SetCellHasCheckBox( vtItem, vtColumn, TRUE );
items.SetCellState( vtItem, vtColumn, 1 );
```

The following VB.NET sample adds a check box that's checked to the focused cell:

```
With AxComboBox1.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:

```
axComboBox1.Items.set_CellHasCheckBox(axComboBox1.Items.FocusItem, 0, true);
axComboBox1.Items.set_CellState(axComboBox1.Items.FocusItem, 0, 1);
```

The following VFP sample adds a check box that's checked to the focused cell:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FocusItem
    .CellHasCheckBox( 0, 0 ) = .t.
    .CellState( 0,0 ) = 1
endwith
```

The following sample shows how to change the state for a cell to checked state:

```
ComboBox1.Items.CellState(ComboBox1.Items(0), 0) = 1,
```

The following sample shows how to change the state for a cell to unchecked state:

```
ComboBox1.Items.CellState(ComboBox1.Items(0), 0) = 0,
```

The following sample shows how to change the state for a cell to partial checked state:

```
ComboBox1.Items.CellState(ComboBox1.Items(0), 0) = 2
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long

type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
```

```
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
    .Items.CellBold(.Items(0), 0) = True
```

```
    .Items.CellBold(.Items(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

```
End With
```

property Items.CellStrikeOut([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value that specifies whether the cell's caption should appear in **strikeout**.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption.
Boolean	A boolean expression that indicates whether the cell should appear in strikeout .

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 [CellCaptionFormat](#) 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 caption of the cell that has the focus:

```
With ComboBox1.Items
    .CellStrikeOut(.FocusItem, 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_combobox.GetItems();
items.SetCellStrikeOut( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ), TRUE );
```

The following C# sample draws a horizontal line through the caption of the cell that has the focus:

```
axComboBox1.Items.set_CellStrikeOut(axComboBox1.Items.FocusItem, 0, true);
```

The following VB.NET sample draws a horizontal line through the caption of the cell that has the focus:

```
With AxComboBox1.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.ComboBox1.Items
    .DefaultItem = .FocusItem
    .CellStrikeOut(0, 0) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
End With
```

property Items.CellToolTip([Item as Variant], [ColIndex as Variant]) as String

Retrieves or sets a value that indicates the cell's tool tip text.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
String	A string expression that indicates the cell's tooltip.

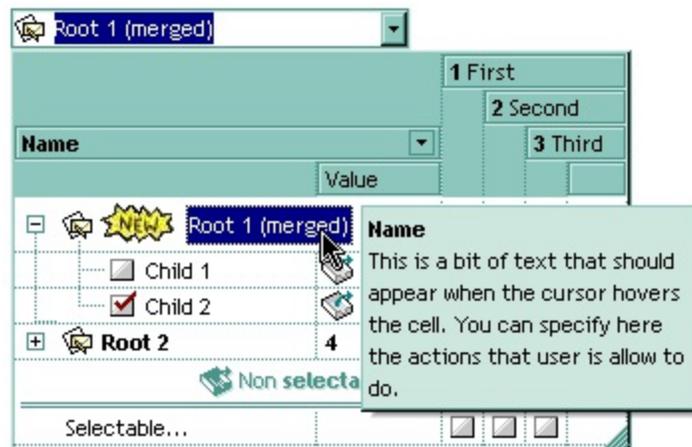
By default, the CellToolTip property is "..." (three dots). If the CellToolTip property is "..." the control displays the cell's caption if it doesn't fit the cell's client area. If the CellToolTip property is different than "...", the control shows a tooltip that displays the CellToolTip value. The control fires the [ToolTip](#) event when the column's tooltip is about to be displayed. 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. The [ToolTipFont](#) property changes the font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color. Use the [ShowToolTip](#) method to display a custom tooltip.

The tooltip supports the following HTML tags:

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

- `key[:width]` inserts a custom size picture being loaded using the [HTMLPicture](#) property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used
- `text ` 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.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout** ("), **&#number**, For instance, the `€` displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`;



Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, that refers a cell.

property Items.CellUnderline([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value that indicates whether the cell is underlined.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell 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 [CellCaptionFormat](#) 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 ComboBox1.Items  
    .CellUnderline(.FocusItem, 0) = True  
End With
```

The following C++ sample underlines the focused cell:

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

The following C# sample underlines the focused cell:

```
axComboBox1.Items.set_CellUnderline(axComboBox1.Items.FocusItem, 0, true);
```

The following VB.NET sample underlines the focused cell:

```
With AxComboBox1.Items  
    .CellUnderline(.FocusItem, 0) = True  
End With
```

The following VFP sample underlines the focused cell:

```
with thisform.ComboBox1.Items  
  .DefaultItem = .FocusItem  
  .CellUnderline(0, 0 ) = .t.  
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

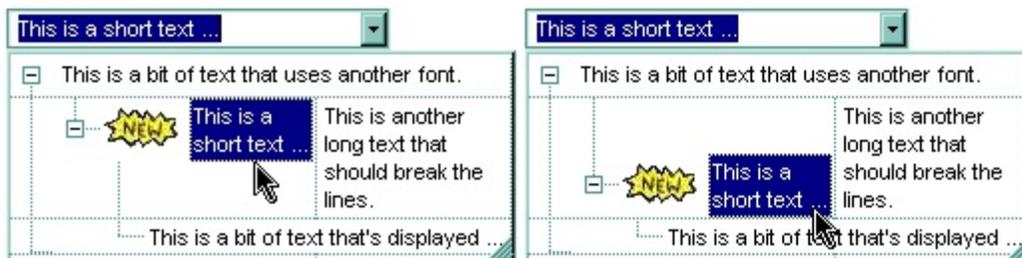
```
With ComboBox1  
  .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True  
  .Items.CellBold(.Items(0), 0) = True  
  .Items.CellBold(.Items(0)) = True  
  .Items.CellBold(.Items.ItemByIndex(0)) = True  
  .Items.CellBold(.Items.ItemByIndex(0), 0) = True  
  .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True  
End With
```

property Items.CellVAlignment ([Item as Variant], [ColIndex as Variant]) as VAlignmentEnum

Retrieves or sets a value that indicates how the cell's caption is vertically aligned.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
VAlignmentEnum	A VAlignmentEnum expression that indicates the cell's vertical alignment.

The CellVAlignment property aligns vertically the cell. The CellVAlignment property aligns the +/- sign if the item contains child items. The CellVAlignment property has effect if the item displays cells using multiple lines. Use the [CellSingleLine](#) property to wrap the cell's caption on multiple lines. Use the [ItemHeight](#) property to specify the height of the item. Use the **
** built-in HTML format to break a line, when [CellCaptionFormat](#) property is exHTML. Use the [CellHAlignment](#) property to align horizontally the cell. The +/- button is aligned accordingly to the cell's caption. 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 ComboBox1.Items
    .CellVAlignment(.FocusItem, 0) = VAlignmentEnum.exBottom
End With
```

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

```
#include "Items.h"
CItems items = m_combobox.GetItems();
items.SetCellVAlignment( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ), 2
/*exBottom*/ );
```

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

```
With AxComboBox1.Items
    .CellVAlignment(.FocusItem, 0) = EXCOMBOBOXLib.VAlignmentEnum.exBottom
End With
```

The following C# sample right aligns the focused cell:

```
axComboBox1.Items.set_CellVAlignment(axComboBox1.Items.FocusItem, 0,
EXCOMBOBOXLib.VAlignmentEnum.exBottom);
```

The following VFP sample right aligns the focused cell:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FocusItem
    .CellVAlignment(0,0) = 2 && exBottom
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With ComboBox1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
End With
```

property Items.ChildCount (Item as HITEM) as Long

Retrieves the number of children items.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long value that indicates the number of child items.

Use the ChildCount property checks whether an item has child items. Use the [ItemChild](#) property to get the first child item, if there is one, 0 else. Use the [ExpandItem](#) property to expand or collapse an item. Use the [ItemCount](#) property to get the number of items in the control. Use the [ItemHasChildren](#) property to specify whether the item should display a +/- sign even if it contains no child items.

method Items.ClearCellBackColor ([Item as Variant], [ColIndex as Variant])

Clears the cell's background color.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
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 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.

method Items.ClearCellForeColor ([Item as Variant], [ColIndex as Variant])

Clears the cell's foreground color.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
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 ClearCellForeColor method clears the cell's foreground color when [CellForeColor](#) property was used. Use the [ForeColor](#) property to specify the control's foreground color.

method Items.ClearCellHAlignment ([Item as Variant], [ColIndex as Variant])

Clears the cell's alignment.

Type	Description
Item as Variant	A long expression that indicates the handle 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` (Item as `HITEM`)

Clears the item's background color.

Type	Description
Item as <code>HITEM</code>	A long expression that indicates the item's handle. If the Item is 0, the <code>ClearItemBackColor</code> clears the background color for all items.

The `ClearItemBackColor` method clears the item's background color when [ItemBackColor](#) property was used. Use the [BackColor](#) property to specify the control's background color.

method Items.ClearItemForeColor (Item as HITEM)

Clears the item's foreground color.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.

The ClearItemForeColor method clears the item's foreground color when [ItemForeColor](#) property was used. Use the [ForeColor](#) property to specify the control's foreground color.

property Items.DefaultItem as HITEM

Retrieves or sets a value that indicates the handle of the item used by Items properties in VFP.

Type	Description
HITEM	Specifies a long expression that indicates the handle of the item that's used by all properties of Items object, that have a parameter Item.

The property is used in VFP implementation. The VFP fires "Invalid Subscript Range" error, while it tries to process a number greater than 65000. Since, the HITEM is a long value that most of the time exceeds 65000, the VFP users have to use this property, instead passing directly the handles to properties. The following sample shows to change the cell's image:

```
.Items.DefaultItem = .Items.AddItem("Item 1")  
.Items.CellImage(0,1) = 2
```

In VFP the following sample fires: "Invalid Subscript Range":

```
i = .Items.AddItem("Item 1")  
.Items.CellImage(i,1) = 2
```

because the i variable is grater than 65000.

So, if you pass zero to a property that has a parameter titled Item, the control takes instead the DefaultItem value.

property Items.EnableItem(Item as HITEM) as Boolean

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

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item is enabled or disabled.

Once that an item was disabled all the cells of that item are disabled, so the [CellEnabled](#) property has no effect. Use the [Enabled](#) property to disable a column. The [SelectableItem](#) property specifies whether the user can select 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.

method Items.EnsureVisibleItem (Item as HITEM)

Ensures that the given item is in the visible client area.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.

The method doesn't expand parent items. The EnsureVisibleItem method scrolls the control's content until the item is visible. Use the [IsItemVisible](#) to check if an item fits the control's client area.

The following VB sample ensures that first item is visible:

```
ComboBox1.Items.EnsureVisibleItem ComboBox1.Items(0)
```

The following C++ sample ensures that first item is visible:

```
#include "Items.h"  
CItems items = m_combobox.GetItems();  
items.EnsureVisibleItem( items.GetItemByIndex( 0 ) );
```

The following C# sample ensures that first item is visible:

```
axComboBox1.Items.EnsureVisibleItem(axComboBox1.Items[0]);
```

The following VB.NET sample ensures that first item is visible:

```
AxComboBox1.Items.EnsureVisibleItem(AxComboBox1.Items(0))
```

The following VFP sample ensures that first item is visible:

```
with thisform.ComboBox1.Items  
    .EnsureVisibleItem( .ItemByIndex( 0 ) )  
endwith
```

property Items.ExpandItem(Item as HITEM) as Boolean

Expands, or collapses, the child items of the specified item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle. If the Item is 0, setting the ExpandItem property expands or collapses all items. For instance, the ExpandItem(0) = False, collapses all items, while the ExpandItem(0) = True, expands all items.
Boolean	A boolean expression that indicates whether the item is expanded or collapsed.

Use ExpandItem property to programmatically expand or collapse an item. Use the ExpandItem property to check whether an item is expanded or collapsed. Before expanding/collapsing an item, the control fires the [BeforeExpandItem](#) event. Use the BeforeExpandItem event to cancel expanding/collapsing of an item. After item is expanded/collapsed the control fires the [AfterExpandItem](#) event. The following samples shows how to expand the selected item:

ComboBox1.Items.ExpandItem(ComboBox1.Items.SelectedItem()) = True. The property has no effect if the item has no child items. To check if the item has child items you can use [ChildCount](#) property. Use the [ItemHasChildren](#) property to display a +/- expand sign to the item even if it doesn't contain child items. The [ExpandOnSearch](#) property specifies whether the control expands nodes when incremental searching is on ([AutoSearch](#) property is different than 0) and user types characters when the control has the focus. Use the [InsertItem](#) property to add child items.

The following VB sample programmatically expands the item when the user selects it :

```
Private Sub ComboBox1_SelectionChanged()  
    ComboBox1.Items.ExpandItem(ComboBox1.Items.SelectedItem()) = True  
End Sub
```

The following VB sample expands programmatically the focused item:

```
With ComboBox1.Items  
    .ExpandItem(.FocusItem) = True  
End With
```

The following C++ sample expands programmatically the focused item:

```
#include "Items.h"
```

```
CItems items = m_combobox.GetItems();  
items.SetExpandItem( items.FocusItem(), TRUE );
```

The following VB.NET sample expands programmatically the focused item:

```
AxComboBox1.Items.ExpandItem( AxComboBox1.Items.FocusItem ) = True
```

The following C# sample expands programmatically the focused item:

```
axComboBox1.Items.set_ExpandItem( axComboBox1.Items.FocusItem, true );
```

The following VFP sample expands programmatically the focused item:

```
with thisform.ComboBox1.Items  
  .DefaultItem = .FocusItem  
  .ExpandItem( 0 ) = .t.  
endwith
```

property Items.FindItem (Caption as Variant, [ColIndex as Variant], [StartIndex as Variant]) as HITEM

Finds an item, looking for Caption in ColIndex colum. 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 string expression that indicates the column's caption, or a long expression that indicates the column's index.
StartIndex as Variant	A long value that indicates the index of item from where the searching starts.
HITEM	A long expression that indicates the item's handle that matches the criteria.

Use the FindItem to search for an item. Finds a control's item that matches [CellCaption](#)(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. The searching is case sensitive only if the ASCIIUpper property is empty. Use the [AutoSearch](#) property to enable incremental search feature within the column.

The following VB sample selects the first item that matches "DUMON" on the first column:

```
ComboBox1.Items.SelectItem(ComboBox1.Items.FindItem("DUMON", 0)) = True
```

The following C++ sample finds and selects an item:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
COleVariant vtMissing;
long hFind = items.GetFindItem( COleVariant("King"), COleVariant("LastName"), vtMissing
);
if ( hFind != NULL )
    items.SetSelectItem( hFind, TRUE );
```

The following C# sample finds and selects an item:

```
axComboBox1.Items.set_SelectItem(axComboBox1.Items.get_FindItem("Child 2", 0, 0),
true);
```

The following VB.NET sample finds and selects an item:

```
With AxComboBox1.Items
  Dim iFind As Integer
  iFind = .FindItem("Child 2", 0)
  If Not (iFind = 0) Then
    .SelectItem(iFind) = True
  End If
End With
```

The following VFP sample finds and selects an item:

```
with thisform.ComboBox1.Items
  .DefaultItem = .FindItem("Child 2",0)
  if ( .DefaultItem <> 0 )
    .SelectItem( 0 ) = .t.
  endif
endwith
```

property Items.FindPath (Path as String) as HITEM

Finds an item given its path.

Type	Description
Path as String	A string expression that indicates the item's path.
HITEM	A long expression that indicates the item's handle that matches the criteria.

The FindPath property searches the item on the column [SearchColumnIndex](#). Use the [FullPath](#) property in order to get the item's path. Use the [FindItem](#) to search for an item.

The following VB sample selects the item based on its path:

```
ComboBox1.Items.SelectItem(ComboBox1.Items.FindPath("Files and Folders\Hidden Files and Folders\Do not show hidden files and folder")) = True
```

The following C++ sample selects the item based on its path:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
COleVariant vtMissing;
long hFind = items.GetFindPath( "Files and Folders\\Hidden Files and Folders\\Do not show hidden files and folder" );
if ( hFind != NULL )
    items.SetSelectItem( hFind, TRUE );
```

The following VB.NET sample selects the item based on its path:

```
With AxComboBox1.Items
    Dim iFind As Integer
    iFind = .FindPath("Files and Folders\Hidden Files and Folders\Do not show hidden files and folder")
    If Not (iFind = 0) Then
        .SelectItem(iFind) = True
    End If
End With
```

The following C# sample selects the item based on its path:

```
int iFind = axComboBox1.Items.get_FindPath("Files and Folders\\Hidden Files and  
Folders\\Do not show hidden files and folder");  
if ( iFind != 0 )  
    axComboBox1.Items.set_SelectItem(iFind, true);
```

The following VFP sample selects the item based on its path:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FindPath("Files and Folders\\Hidden Files and Folders\\Do not show  
hidden files and folder")  
    if ( .DefaultItem <> 0 )  
        .SelectItem( 0 ) = .t.  
    endif  
endwith
```

property Items.FirstVisibleItem as HITEM

Retrieves the handle of the first visible item in control.

Type	Description
HITEM	A long expression that indicates the handle of the first visible item.

Use the FirstVisibleItem, [NextVisibleItem](#) and [IsItemVisible](#) properties to get the items that fit the client area. Use the NextVisibleItem property to get the next visible item. Use the IsVisibleItem property to check whether an item fits the control's client area.

The following VB sample enumerates all visible items in the control's list:

```
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = ComboBox1.Columns.Count
With ComboBox1.Items
    h = .FirstVisibleItem
    While Not (h = 0)
        Dim s As String
        s = ""
        For j = 0 To nCols - 1
            s = s + .CellCaption(h, j) + Chr(9)
        Next
        Debug.Print s
        h = .NextVisibleItem(h)
    Wend
End With
```

The following C++ sample enumerates all visible items in the control's list:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
long hItem = items.GetFirstVisibleItem();
while ( hItem && items.GetIsItemVisible( hItem ) )
{
    OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
}
```

```
hltem = items.GetNextVisibleItem( hltem );  
}
```

The following VB.NET sample enumerates all visible items in the control's list:

```
With AxComboBox1.Items  
    Dim hltem As Integer  
    hltem = .FirstVisibleItem  
    While Not (hltem = 0)  
        Debug.Print(.CellCaption(hltem, 0))  
        hltem = .NextVisibleItem(hltem)  
    End While  
End With
```

The following C# sample enumerates all visible items in the control's list:

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;  
int hltem = items.FirstVisibleItem;  
while (hltem != 0)  
{  
    object strCaption = items.get_CellCaption(hltem, 0);  
    System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");  
    hltem = items.get_NextVisibleItem(hltem);  
}
```

The following VFP sample all visible items in the control's list:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FirstVisibleItem  
    do while ( .DefaultItem <> 0 )  
        wait window .CellCaption( 0, 0 )  
        .DefaultItem = .NextVisibleItem( 0 )  
    enddo  
endwith
```

property Items.FocusItem as HITEM

Retrieves the handle of item that has the focus.

Type	Description
HITEM	A long expression that indicates the handle of 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 [SelectedItem](#) property to get the selected item. Use the [SelectItem](#) to select or unselect a specified item. Use the [ShowFocusRect](#) property to indicate whether the control draws a marking rectangle around the focused item. Use the [SelectableItem](#) property to specify whether an item is selectable/focusable.

property Items.FormatCell([Item as Variant], [ColIndex as Variant]) as String

Specifies the custom format to display the cell's content.

Type	Description
Item as Variant	A long expression that indicates the handle 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 [CellCaption](#) 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 "**Mihai** Filimon".

- the "`len(value) ? ((0:=dbl(value)) < 10 ? '<fgcolor=808080>' : '') + 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
Root				
Child 1	7+	3+	1=	\$11.00
Child 2	2+	6+	12=	\$19.00
Child 3	2+	2+	4=	\$8.00
Child 4	2+	9+	4=	\$15.00

The **value** keyword in the FormatColumn 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 [CellCaption](#) 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 ""`

Col 1	Col 2
1	<input type="checkbox"/> Root A
4	<input type="checkbox"/> Root B
5	<input type="checkbox"/> Child 1
6	<input type="checkbox"/> Child 2

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

Col 1	Col 2
A	+ Root A
D	- Root B
E	Child 1
F	Child 2

- 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 ""`

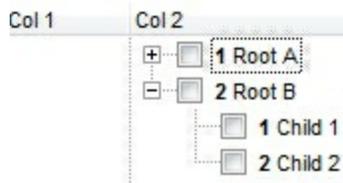
Col 1	Col 2
1	+ Root A
2	- Root B
3	Child 1
4	Child 2

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

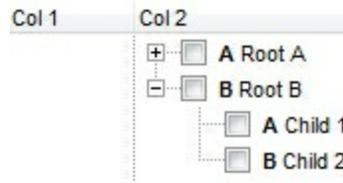
Col 1	Col 2
A	+ Root A
B	- Root B
C	Child 1
D	Child 2

- 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"`

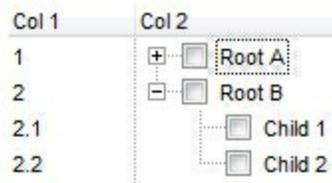


In the following screen shot the `FormatColumn("Col 2") = "' + 1 pos 'A-Z' + '' + value"`

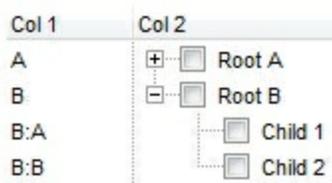


- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance,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

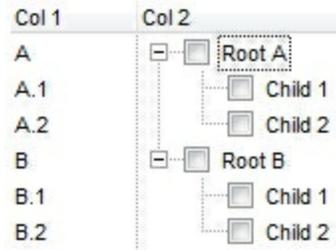
In the following screen shot the `FormatColumn("Col 1") = "1 rpos ""`



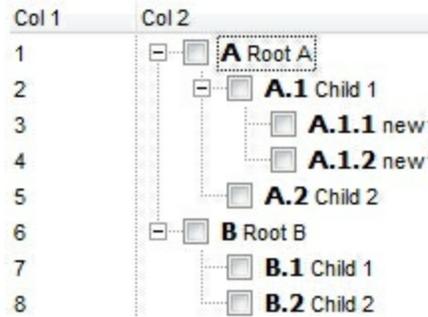
In the following screen shot the `FormatColumn("Col 1") = "1 rpos ':|A-Z'"`



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.FullPath (Item as HITEM) as String

Returns the fully qualified path of the referenced item in an ExComboBox control.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
String	A string expression that indicates the fully qualified path.

Use the FullPath property in order to get the fully qualified path of the referenced item. Use [PathSeparator](#) to change the separator used by FullPath property. Use the [FindPath](#) property to get the item's selected based on its path. The fully qualified path is the concatenation of the text in the given cell's caption property on the column [SearchColumnIndex](#) with the [CellCaption](#) property values of all its ancestors.

method Items.InsertItem ([Parent as HITEM], [UserData as Variant], [Caption as Variant])

Inserts a new item, and returns a handle to the newly created item.

Type	Description
Parent as HITEM	An long expression that indicates the handle of the parent item. The parent item must not be a locked item.
UserData as Variant	A Variant expression that indicates the item's extra data.
Caption as Variant	A Variant expression that indicates the item's caption. The Caption supports built-in HTML format if CellCaptionFormat property is exHTML.

Return	Description
HITEM	Retrieves the handle of the newly created item.

Use the InsertItem method to insert child items. Use the [AddItem](#) method to add a new item. The InsertItem method fails, if the control contains no columns. Use the [Add](#) method to add new columns to the control. The [InsertItem](#) event is fired when a new item is inserted. The Parent parameter indicates the handle of the parent item. The UserData parameter indicates the item's user data. The Caption argument indicates the caption of the newly inserted item for the first column. Use the [ItemPosition](#) property to change the item's position. If the control has multiple columns use the [CellCaption](#) property to set caption for the rest of the columns. Use the [PutItems](#) method to load an array of elements. Use the [ItemData](#) property to retrieve the UserData parameter, after item is inserted. Use the [LockedItemCount](#) property to lock or unlock items to the top or bottom side of the control's list. Use the [MergeCells](#) method to combine two or more cells in a single cell. Use the [ItemChild](#) property to get the first child item. Use the [ItemParent](#) property to retrieve the handle of the parent item. Use the [SelectableItem](#) property to specify whether the user can select the item. Use the [LinesAtRoot](#) property to link items at the root of the hierarchy. The exComputedField type specifies a computed cell.

The following VB sample loads some items using safe arrays:

```
With ComboBox1
    .BeginUpdate

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

With .Items

```
Dim h As HITEM
```

```
h = .AddItem(Array("Item 1", "Item 2", "Item 3"))
```

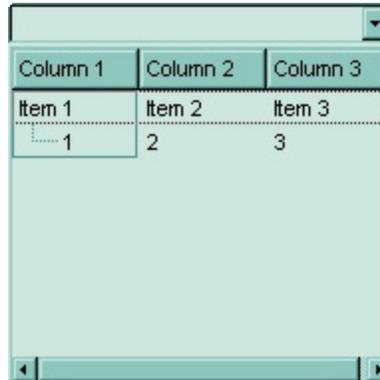
```
.InsertItem h, , Array(1, 2, 3)
```

```
.ExpandItem(h) = True
```

End With

```
.EndUpdate
```

End With



Column 1	Column 2	Column 3
Item 1	Item 2	Item 3
1	2	3

The following VB sample adds items to a single column control:

With ComboBox1

```
.BeginUpdate
```

```
.ColumnAutoSize = True
```

```
.HeaderVisible = False
```

```
.LinesAtRoot = exLinesAtRoot
```

```
.HasButtons = exCircle
```

```
.Columns.Add "Column 1"
```

With .Items

```
Dim h As HITEM
```

```
h = .AddItem("Item 1")
```

```
.InsertItem h, , "Item 1.1"
```

```
.InsertItem h, , "Item 1.2"
```

```
.ExpandItem(h) = True
```

End With

```
.EndUpdate
```

```
End With
```

The following VB sample inserts a child item and expands the focused item:

```
With ComboBox1.Items
```

```
    .InsertItem .FocusItem, , "new child"
```

```
    .ExpandItem(.FocusItem) = True
```

```
End With
```

The following C++ sample inserts a child item and expands the focused item:

```
#include "Items.h"
```

```
CItems items = m_combobox.GetItems();
```

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
```

```
long h = items.InsertItem( items.GetFocusItem(), vtMissing, COleVariant( "new child" ) );
```

```
items.SetExpandItem( items.GetFocusItem(), TRUE );
```

The following VB.NET sample inserts a child item and expands the focused item:

```
With AxComboBox1.Items
```

```
    Dim hItem As Integer = .InsertItem(.FocusItem, , "new child")
```

```
    .ExpandItem(.FocusItem) = True
```

```
End With
```

The following C# sample inserts a child item and expands the focused item:

```
int hItem = axComboBox1.Items.InsertItem(axComboBox1.Items.FocusItem, null, "new  
child");
```

```
axComboBox1.Items.set_ExpandItem(axComboBox1.Items.FocusItem, true);
```

The following VFP sample inserts a child item and expands the focused item:

```
with thisform.ComboBox1.Items
```

```
    .DefaultItem = .InsertItem( .FocusItem, "", "new child" )
```

```
    .DefaultItem = .FocusItem
```

```
    .ExpandItem(0) = .t.
```

```
endwith
```

property Items.IsItemLocked (Item as HITEM) as Boolean

Returns a value that indicates whether the item is locked or unlocked. */*not supported in the lite version*/*

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Boolean	A boolean expression that indicates whether the item is locked or unlocked.

Use the IsItemLocked property to check whether an item is locked or unlocked. A locked item is always displayed on the top or bottom side of the control no matter if the control's list is scrolled up or down. Use the [LockedItemCount](#) property to add or remove items fixed/locked to the top or bottom side of the control. Use the [LockedItem](#) property to access a locked item by its position. Use the [ShowLockedItems](#) property to show or hide the locked items.

The following VB sample prints the locked item from the cursor:

```
Private Sub ComboBox1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    On Error Resume Next
    ' Converts the container coordinates to client coordinates
    X = X / Screen.TwipsPerPixelX
    Y = Y / Screen.TwipsPerPixelY
    Dim h As HITEM
    Dim c As Long
    Dim hit As EXCOMBOBOXLibCtl.HitTestInfoEnum
    ' Gets the item from (X,Y)
    With ComboBox1
        h = .ItemFromPoint(X, Y, c, hit)
        If Not (h = 0) Then
            If (.Items.IsItemLocked(h)) Then
                Debug.Print .Items.CellCaption(h, c)
            End If
        End If
    End With
End Sub
```

The following C++ sample prints the locked item from the cursor:

```

#include "Items.h"
void OnMouseMoveComboBox1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, hItem = m_combobox.GetItemFromPoint( X, Y, &c, &hit );
    if ( hItem != 0 )
    {
        CItems items = m_combobox.GetItems();
        if ( items.GetIsItemLocked( hItem ) )
        {
            COleVariant vtItem( hItem ), vtColumn( c );
            CString strCaption = V2S( &items.GetCellCaption( vtItem, vtColumn ) ), strOutput;
            strOutput.Format( "Cell: '%s', Hit = %08X\n", strCaption, hit );
            OutputDebugString( strOutput );
        }
    }
}

```

The following VB.NET sample prints the locked item from the cursor:

```

Private Sub AxComboBox1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent) Handles
AxComboBox1.MouseMoveEvent
    With AxComboBox1
        Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If Not (i = 0) Then
            With .Items
                If (.IsItemLocked(i)) Then
                    Debug.WriteLine("Cell: " & .CellCaption(i, c) & " Hit: " & hit.ToString())
                End If
            End With
        End If
    End With
End Sub

```

The following C# sample prints the locked item from the cursor:

```

private void axComboBox1_MouseMoveEvent(object sender,

```

```

AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent e)
{
    int c = 0;
    EXCOMBOBOXLib.HitTestInfoEnum hit;
    int i = axComboBox1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i != 0)
        if ( axComboBox1.Items.get_IsItemLocked( i ) )
            {
                object cap = axComboBox1.Items.get_CellCaption(i, c);
                string s = cap != null ? cap.ToString() : "";
                s = "Cell: " + s + ", Hit: " + hit.ToString();
                System.Diagnostics.Debug.WriteLine(s);
            }
}
}

```

The following VFP sample prints the locked item from the cursor:

```

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

local c, hit
c = 0
hit = 0
with thisform.ComboBox1
    .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
    with .Items
        if ( .DefaultItem <> 0 )
            if ( .IsItemLocked( 0 ) )
                wait window nowait .CellCaption( 0, c ) + " " + Str( hit )
            endif
        endif
    endwhile
endwith

```

property Items.IsItemVisible (Item as HITEM) as Boolean

Checks if the specific item fits the control's client area.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item fits the client area.

To make sure that an item fits the client area call the [EnsureVisibleItem](#) method. Use the [FirstVisibleItem](#), [NextVisibleItem](#) and `IsItemVisible` properties to get the items that fit the client area. Use the `NextVisibleItem` property to get the next visible item. Use the `IsVisibleItem` property to check whether an item fits the control's client area.

property Items.ItemAllowSizing(Item as HITEM) as Boolean

Retrieves or sets a value that indicates whether a user can resize the item at run-time.

Type	Description
Item as HITEM	A HITEM expression that indicates the handle 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(Item as HITEM) as Color

Retrieves or sets a background color for a specific item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle. If the Item is 0, the ItemBackColor changes the background color for all items.
Color	A color expression that indicates the item'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 change the cell's background color. To change the background color of the entire control you can call [BackColor](#) property of the control. Use the [ClearItemBackColor](#) property to clear the item's background color, after setting using the ItemBackColor property. For instance, you can create a transparent skin and put on the item's background. Use the [Add](#) method to add new skins 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.

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:

```
axComboBox1.Items.set_ItemBackColor(axComboBox1.Items.FocusItem,  
ToUInt32(Color.Red) );
```

The following VB.NET sample changes the background color for the focused item:

```
With AxComboBox1.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_combobox.GetItems();  
items.SetItemBackColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the background color for the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .ItemBackColor( 0 ) = RGB(255,0,0)  
endwith
```

property Items.ItemBold(Item as HITEM) as Boolean

Retrieves or sets a value that indicates whether the item is bolded.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the items 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 [CellCaptionFormat](#) 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 hOldBold As HITEM
```

```
Private Sub ComboBox1_SelectionChanged()  
    If Not (hOldBold = 0) Then  
        ComboBox1.Items.ItemBold(hOldBold) = False  
    End If  
    hOldBold = ComboBox1.Items.SelectedItem()  
    ComboBox1.Items.ItemBold(hOldBold) = True  
End Sub
```

The following VB sample bolds the focused item:

```
With ComboBox1.Items  
    .ItemBold(.FocusItem) = True  
End With
```

The following C++ sample bolds the focused item:

```
#include "Items.h"  
CItems items = m_combobox.GetItems();  
items.SetItemBold( items.GetFocusItem() , TRUE );
```

The following C# sample bolds the focused item:

```
axComboBox1.Items.set_ItemBold(axComboBox1.Items.FocusItem, true);
```

The following VB.NET sample bolds the focused item:

```
With AxComboBox1.Items  
    .ItemBold(.FocusItem) = True  
End With
```

The following VFP sample bolds the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .ItemBold( 0 ) = .t.  
endwith
```

property Items.ItemByIndex (nIndex as Long) as HITEM

Retrieves the handle of the item given its index in the Items collection.

Type	Description
nIndex as Long	A long value that indicates the item's index.
HITEM	A long expression that indicates the item's handle.

Use the `ItemByIndex` to get the index of an item. Use the [ItemCount](#) property to count the items in the control. Use the [ItemPosition](#) property to get the item's position. Use the [ItemToIndex](#) property to get the index of giving item. For instance, The `ItemByIndex` property is the default property for Items object, so the following statements are equivalents: `ComboBox1.Items(0)`, `ComboBox1.Items.ItemByIndex(0)`. Use the [FirstVisibleItem](#) and [NextVisibleItem](#) properties to enumerate the items as they are sorted.

The following VB sample enumerates all items in the control:

```
Dim i As Long, n As Long
With ComboBox1.Items
    n = .ItemCount
    For i = 0 To n - 1
        Debug.Print .ItemByIndex(i)
    Next
End With
```

The following C++ sample enumerates all items in the control:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
for ( long i = 0; i < items.GetItemCount(); i++ )
{
    COleVariant vtItem( items.GetItemByIndex( i ) ), vtColumn( long(0) );
    CString strCaption = V2S( &items.GetCellCaption( vtItem, vtColumn ) ), strOutput;
    strOutput.Format( "Cell: '%s'\n", strCaption );
    OutputDebugString( strOutput );
}
```

The following VB.NET sample enumerates all items in the control:

```
With AxComboBox1
```

```
    Dim i As Integer
```

```
    For i = 0 To .Items.ItemCount - 1
```

```
        Debug.Print(.Items.CellCaption(.Items(i), 0))
```

```
    Next
```

```
End With
```

The following C# sample enumerates all items in the control:

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;
```

```
for (int i = 0; i < items.ItemCount; i++)
```

```
{
```

```
    object caption = items.get_CellCaption(items[i], 0);
```

```
    string strCaption = caption != null ? caption.ToString() : "";
```

```
    System.Diagnostics.Debug.WriteLine(strCaption);
```

```
}
```

The following VFP sample enumerates all items in the control:

```
with thisform.ComboBox1.Items
```

```
    local i
```

```
    for i = 0 to .ItemCount - 1
```

```
        .DefaultItem = .ItemByIndex( i )
```

```
        wait window nowait .CellCaption(0,0)
```

```
    next
```

```
endwith
```

property Items.ItemCell (Item as HITEM, ColIndex as Variant) as HCELL

Retrieves the cell's handle given the item and the column.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the the column's index, a string expression that indicates the column's caption or the column's key.
HCELL	A long value that indicates the cell's handle.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of [Items](#) object that have two parameters *Item* and *ColIndex*, refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

With ComboBox1

```
.Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
```

```
.Items.CellBold(.Items(0), 0) = True
```

```
.Items.CellBold(.Items(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
.Items.CellBold(.Items.ItemByIndex(0), 0) = True
```

```
.Items.CellBold(.Items(0), ComboBox1.Columns(0).Caption) = True
```

End With

property Items.ItemChild (Item as HITEM) as HITEM

Retrieves the first child item of a specified item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the handle of the first child item.

If the ItemChild property gets 0, the item has no child items. Use the ItemChild property to get the first child of an item. The [NextVisibleItem](#) or [NextSiblingItem](#) gets the next visible, sibling item. Use the [InsertItem](#) method to insert a child item. Use the [ItemParent](#) property to retrieve the handle of the parent item. The control displays a +/- sign to parent items, if the [HasButtons](#) property is not zero, the ItemChild property is not empty, or the [ItemHasChildren](#) property is True.

The following VB function recursively enumerates the item and all its child items:

```
Sub Recltem(ByVal c As EXCOMBOBOXLibCtl.ComboBox, ByVal h As HITEM)
  If Not (h = 0) Then
    Dim hChild As HITEM
    With c.Items
      Debug.Print .CellCaption(h, 0)
      hChild = .ItemChild(h)
      While Not (hChild = 0)
        Recltem c, hChild
        hChild = .NextSiblingItem(hChild)
      Wend
    End With
  End If
End Sub
```

The following C++ function recursively enumerates the item and all its child items:

```
void Recltem( CComboBox* pComboBox, long hItem )
{
  COleVariant vtColumn( (long)0 );
  if ( hItem )
  {
    CItems items = pComboBox->GetItems();
```

```

    CString strCaption = V2S( &items.GetCellCaption( COleVariant( hltem ), vtColumn ) ),
strOutput;
    strOutput.Format( "Cell: '%s'\n", strCaption );
    OutputDebugString( strOutput );

    long hChild = items.GetItemChild( hltem );
    while ( hChild )
    {
        Recltem( pComboBox, hChild );
        hChild = items.GetNextSiblingItem( hChild );
    }
}
}

```

The following VB.NET function recursively enumerates the item and all its child items:

```

Shared Sub Recltem(ByVal c As AxEXCOMBOBOXLib.AxComboBox, ByVal h As Integer)
    If Not (h = 0) Then
        Dim hChild As Integer
        With c.Items
            Debug.WriteLine(.CellCaption(h, 0))
            hChild = .ItemChild(h)
            While Not (hChild = 0)
                Recltem(c, hChild)
                hChild = .NextSiblingItem(hChild)
            End While
        End With
    End If
End Sub

```

The following C# function recursively enumerates the item and all its child items:

```

internal void Recltem(AxEXCOMBOBOXLib.AxComboBox combobox, int hltem)
{
    if (hltem != 0)
    {
        EXCOMBOBOXLib.Items items = combobox.Items;
    }
}

```

```

object caption = items.get_CellCaption( hItem, 0 );
System.Diagnostics.Debug.WriteLine(caption != null ? caption.ToString() : "");

int hChild = items.get_ItemChild(hItem);
while (hChild != 0)
{
    RecItem(combobox, hChild);
    hChild = items.get_NextSiblingItem(hChild);
}
}
}
}

```

The following VFP function recursively enumerates the item and all its child items (recitem method):

```

LPARAMETERS h

with thisform.ComboBox1
  If ( h != 0 ) Then
    local hChild
    With .Items
      .DefaultItem = h
      wait window .CellCaption(0, 0)
      hChild = .ItemChild(h)
      do While (hChild != 0)
        thisform.recitem(hChild)
        hChild = .NextSiblingItem(hChild)
      enddo
    EndWith
  EndIf
endwith

```

property Items.ItemCount as Long

Retrieves the number of items.

Type	Description
Long	A long value that indicates the number of items into Items collection.

The ItemCount property gets the number of items in the control. Use the [VisibleItemCount](#) property to specify the number of visible items in the list. Use the [ItemByIndex](#) property to get the handle of the item giving its index. Use [ChildCount](#) to get the number of child items giving an item. Use the [ItemChild](#) property to get the first child item. Use the [FirstVisibleItem](#) property to get the first visible item. Use the [NextVisibleItem](#) property to get the next visible item. The [NextSiblingItem](#) property retrieves the next sibling of the item in the parent's child list. Use the [ItemPosition](#) property to change the item's position. Use the [AddItem](#), [InsertItem](#), [PutItems](#) or [DataSource](#) property to add new items to the control.

The following VB sample enumerates all items in the control:

```
Dim i As Long, n As Long
With ComboBox1.Items
    n = .ItemCount
    For i = 0 To n - 1
        Debug.Print .ItemByIndex(i)
    Next
End With
```

The following C++ sample enumerates all items in the control:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
for ( long i = 0; i < items.GetItemCount(); i++ )
{
    COleVariant vtItem( items.GetItemByIndex( i ) ), vtColumn( long(0) );
    CString strCaption = V2S( &items.GetCellCaption( vtItem, vtColumn ) ), strOutput;
    strOutput.Format( "Cell: '%s'\n", strCaption );
    OutputDebugString( strOutput );
}
```

The following VB.NET sample enumerates all items in the control:

```
With AxComboBox1
  Dim i As Integer
  For i = 0 To .Items.ItemCount - 1
    Debug.Print(.Items.CellCaption(.Items(i), 0))
  Next
End With
```

The following C# sample enumerates all items in the control:

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;
for (int i = 0; i < items.ItemCount; i++)
{
  object caption = items.get_CellCaption(items[i], 0);
  string strCaption = caption != null ? caption.ToString() : "";
  System.Diagnostics.Debug.WriteLine(strCaption);
}
```

The following VFP sample enumerates all items in the control:

```
with thisform.ComboBox1.Items
  local i
  for i = 0 to .ItemCount - 1
    .DefaultItem = .ItemByIndex( i )
    wait window nowait .CellCaption(0,0)
  next
endwith
```

property Items.ItemData(Item as HITEM) as Variant

Retrieves or sets the extra data for a specific item.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Variant	A variant value 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.

property Items.ItemDivider(Item as HITEM) as Long

Specifies whether the item acts like a divider or normal item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long expression that indicates the column's index.

A divider item uses the item's client area to display a single cell. The ItemDivider property specifies the index of the cell being displayed. In other words, the divider item merges the item cells into a single cell. Use the [ItemDividerLine](#) property to define the line that underlines the divider item. Use the [LockedItemCount](#) property to lock items on the top or bottom side of the control. Use the [MergeCells](#) method to combine two or multiple cells in a single cell. Use the [SelectableItem](#) property to specify the user can select an item. A divider item has sense for a control with multiple columns.

The following VB sample adds a divider item that's locked to the top side of the control (Before running this sample please make sure that your control has columns):

With ComboBox1

.BeginUpdate

.DrawGridLines = exNoLines

With .Items

.LockedItemCount(TopAlignment) = 1

Dim h As HITEM

h = .LockedItem(TopAlignment, 0)

.ItemDivider(h) = 0

.ItemHeight(h) = 22

.CellCaption(h, 0) = "Total:"

\$12.344.233"

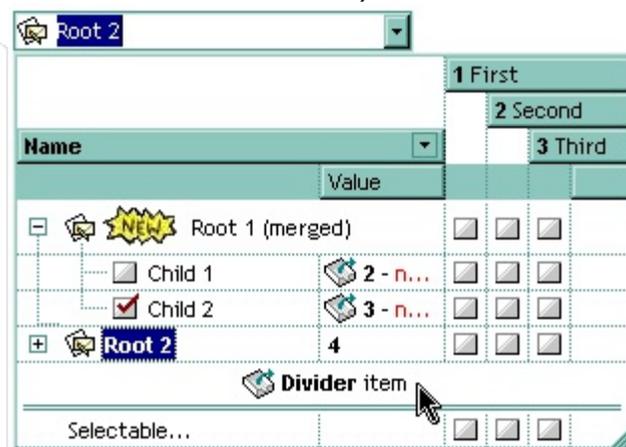
.CellCaptionFormat(h, 0) = exHTML

.CellHAlignment(h, 0) = RightAlignment

End With

.EndUpdate

End With



The following C++ sample adds a divider item, that's not selectable too:

```
#include "Items.h"
```

```
Cltems items = m_combobox.GetItems();
```

```
long i = items.AddItem( COleVariant("divider item") );  
items.SetItemDivider( i, 0 );  
items.SetSelectableItem( i, FALSE );
```

The following C# sample adds a divider item, that's not selectable too:

```
int i = axComboBox1.Items.AddItem("divider item");  
axComboBox1.Items.set_ItemDivider(i, 0);  
axComboBox1.Items.set_SelectableItem(i, false);
```

The following VB.NET sample adds a divider item, that's not selectable too:

```
With AxComboBox1.Items  
    Dim i As Integer  
    i = .AddItem("divider item")  
    .ItemDivider(i) = 0  
    .SelectableItem(i) = False  
End With
```

The following VFP sample adds a divider item, that's not selectable too:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .AddItem("divider item")  
    .ItemDivider(0) = 0  
    .SelectableItem(0) = .f.  
endwith
```

property Items.ItemDividerLine(Item as HITEM) as DividerLineEnum

Defines the type of line in the divider item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
DividerLineEnum	A DividerLineEnum expression that indicates the type of the line in the divider item.

Use the [ItemDivider](#) property to define a divider item. Use the ItemDividerLine and [ItemDividerAlignment](#) properties to define the style of the line into a divider item. Use the [MergeCells](#) method to combine two or more cells in a single cell. Use the [LockedItemCount](#) property to add or remove items fixed/locked to the top or bottom side of the control.

property Items.ItemDividerLineAlignment(Item as HITEM) as DividerAlignmentEnum

Specifies the alignment of the line in the divider item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle
DividerAlignmentEnum	A DividerAlignmentEnum expression that specifies the line's alignment.

By default, the ItemDividerLineAlignment property is DividerBottom. The Use the [ItemDividerLine](#) and ItemDividerLineAlignment properties to define the style of the line into a divider item. Use the [ItemDivider](#) property to define a divider item.

property Items.ItemFont (Item as HITEM) as IFontDisp

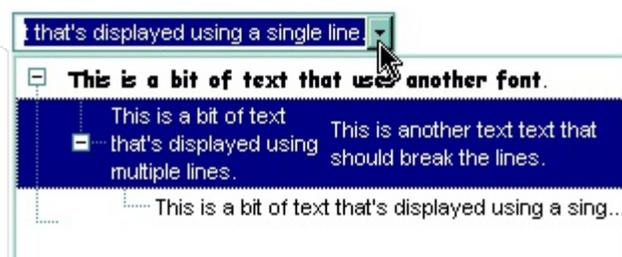
Retrieves or sets the item's font.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
IFontDisp	A font object being used.

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 [CellCaptionFormat](#) to specify different font attributes. Use the [ItemHeight](#) property to specify the height of the item. 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:

```
ComboBox1.BeginUpdate
With ComboBox1.Items
    .ItemFont(.FocusItem) = ComboBox1.Font
    With .ItemFont(.FocusItem)
        .Name = "Comic Sans MS"
        .Bold = True
    End With
End With
ComboBox1.EndUpdate
```



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

```
#include "Items.h"
#include "Font.h"
CItems items = m_combobox.GetItems();
m_combobox.BeginUpdate();
items.SetItemFont( items.GetFocusItem(), m_combobox.GetFont().m_lpDispatch );
COleFont font = items.GetItemFont( items.GetFocusItem() );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_combobox.EndUpdate();
```

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

```
AxComboBox1.BeginUpdate()  
With AxComboBox1.Items  
    .ItemFont(.FocusItem) = IFDH.GetIFontDisp(AxComboBox1.Font)  
    With .ItemFont(.FocusItem)  
        .Name = "Comic Sans MS"  
        .Bold = True  
    End With  
End With  
AxComboBox1.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:

```
axComboBox1.BeginUpdate();  
axComboBox1.Items.set_ItemFont(axComboBox1.Items.FocusItem,  
IFDH.GetIFontDisp(axComboBox1.Font));  
stdole.IFontDisp spFont =  
axComboBox1.Items.get_ItemFont(axComboBox1.Items.FocusItem);  
spFont.Name = "Comic Sans MS";  
spFont.Bold = true;  
axComboBox1.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.ComboBox1.BeginUpdate()
```

```
with thisform.ComboBox1.Items
```

```
    .DefaultItem = .FocusItem
```

```
    .ItemFont(0) = thisform.ComboBox1.Font
```

```
with .ItemFont(0)
```

```
    .Name = "Comic Sans MS"
```

```
    .Bold = .t.
```

```
endwith
```

```
endwith
```

```
thisform.ComboBox1.EndUpdate()
```

property Items.ItemForeColor(Item as HITEM) as Color

Retrieves or sets a foreground color for a specific item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Color	A color expression that defines the item's foreground color.

Use the [CellForeColor](#) property to change the item's foreground color. Use the [ForeColor](#) property to change the control's foreground color. Use the [ClearItemForeColor](#) property to clear the item's foreground color. The [SelectableItem](#) property specifies whether the user can select an item. 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 cells in the first column as user add new items:

```
Private Sub ComboBox1_AddItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    ComboBox1.Items.CellForeColor(Item, 0) = vbBlue
End Sub
```

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:

```
axComboBox1.Items.set_ItemForeColor(axComboBox1.Items.FocusItem,  
ToUInt32(Color.Red) );
```

The following VB.NET sample changes the foreground color of the focused item:

```
With AxComboBox1.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_combobox.GetItems();  
items.SetItemForeColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the foreground color of the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .ItemForeColor( 0 ) = RGB(255,0,0)  
endwith
```

property Items.ItemHasChildren (Item as HITEM) as Boolean

Adds an expand button to left side of the item even if the item has no child items.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Boolean	A boolean expression that indicates whether the control adds an expand button to the left side of the item even if the item has no child items.

By default, the ItemHasChildren property is False. Use the ItemHasChildren property to build a virtual combobox. Use the [BeforeExpandItem](#) event to add new child items to the expanded item. Use the [ItemChild](#) property to get the first child item, if exists. Use the ItemChild or [ChildCount](#) property to determine whether an item contains child items. The control displays a +/- sign to parent items, if the [HasButtons](#) property is not empty, the ItemChild property is not empty, or the ItemHasChildren property is True. Use the [InsertItem](#) method to insert a new child item. Use the [CellData](#) or [ItemData](#) property to assign an extra value to a cell or to an item.

The following VB sample inserts a child item as soon as user expands an item (the sample has effect only if your control contains items that have the ItemHasChildren property on True):

```
Private Sub ComboBox1_BeforeExpandItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM,
Cancel As Variant)
    With ComboBox1.Items
        If (.ItemHasChildren(Item)) Then
            If .ChildCount(Item) = 0 Then
                Dim h As Long
                h = .InsertItem(Item, , "new " & Item)
            End If
        End If
    End With
End Sub
```

The following VB.NET sample inserts a child item when the user expands an item that has the ItemHasChildren property on True:

```
Private Sub AxComboBox1_BeforeExpandItem(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent) Handles
```

```

AxComboBox1.BeforeExpandItem
  With AxComboBox1.Items
    If (.ItemHasChildren(e.item)) Then
      If .ChildCount(e.item) = 0 Then
        Dim h As Long
        h = .InsertItem(e.item, , "new " & e.item.ToString())
      End If
    End If
  End With
End Sub

```

The following C# sample inserts a child item when the user expands an item that has the ItemHasChildren property on True:

```

private void axComboBox1_BeforeExpandItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent e)
{
  EXCOMBOBOXLib.Items items = axComboBox1.Items;
  if ( items.get_ItemHasChildren( e.item ) )
    if (items.get_ChildCount(e.item) == 0)
      {
        items.InsertItem(e.item, null, "new " + e.item.ToString());
      }
}

```

The following C++ sample inserts a child item when the user expands an item that has the ItemHasChildren property on True:

```

#include "Items.h"
void OnBeforeExpandItemComboBox1(long Item, VARIANT FAR* Cancel)
{
  CItems items = m_combobox.GetItems();
  if ( items.GetItemHasChildren( Item ) )
    if ( items.GetChildCount( Item ) == 0 )
      {
        COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
        items.InsertItem( Item, vtMissing, COleVariant( "new item" ) );
      }
}

```

```
}
```

The following VFP sample inserts a child item when the user expands an item that has the ItemHasChildren property on True(BeforeExpandItem event):

```
*** ActiveX Control Event ***  
LPARAMETERS item, cancel  
  
with thisform.ComboBox1.Items  
  if ( .ItemHasChildren( item ) )  
    if ( .ChildCount( item ) = 0 )  
      .InsertItem(item,"","new " + trim(str(item)))  
    endif  
  endif  
endwith
```

property Items.ItemHeight(Item as HITEM) as Long

Retrieves or sets the item's height.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item. If the Item is 0, setting the ItemHeight property changes the height for all items. For instance, the ItemHeight(0) = 24, changes the height for all items to be 24 pixels wide.
Long	A long value that indicates the item's height.

To change the default height of the item before inserting it into the items collection you can call [DefaultItemHeight](#) property. The control supports items with different heights. If the [CellSingleLine](#) property is False, 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 [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(Item as HITEM) as Boolean

Retrieves or sets a value that indicates whether the item should appear in italic.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item's font attributes include Italic attribute.

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 [CellCaptionFormat](#) 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 selected item:

```
Private Sub ComboBox1_SelectionChanged()  
    If Not (h = 0) Then ComboBox1.Items.ItemItalic(h) = False  
    h = ComboBox1.Items.SelectedItem()  
    ComboBox1.Items.ItemItalic(h) = True  
End Sub
```

The following VB sample makes italic the focused item:

```
With ComboBox1.Items  
    .ItemItalic(.FocusItem) = True  
End With
```

The following C++ sample makes italic the focused item:

```
#include "Items.h"  
CItems items = m_combobox.GetItems();  
items.SetItemItalic( items.GetFocusItem() , TRUE );
```

The following C# sample makes italic the focused item:

```
axComboBox1.Items.set_ItemItalic(axComboBox1.Items.FocusItem, true);
```

The following VB.NET sample makes italic the focused item:

```
With AxComboBox1.Items  
    .ItemItalic(.FocusItem) = True  
End With
```

The following VFP sample makes italic the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .ItemItalic( 0 ) = .t.  
endwith
```

property Items.ItemMaxHeight(Item as HITEM) as Long

Retrieves or sets a value that indicates the maximum height when the item's height is variable.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item. If the Item is 0, setting the ItemMaxHeight property changes the maximum-height for all items. For instance, the ItemMaxHeight(0) = 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 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(Item as HITEM) as Long

Retrieves or sets a value that indicates the minimum height when the item's height is sizing.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item. If the Item is 0, setting the ItemMinHeight property changes the minimum-height for all items. For instance, the ItemMinHeight(0) = 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.ItemParent (Item as HITEM) as HITEM

Returns the handle of the item's parent item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the item's handle that indicates the parent item.

To change the item's parent call [SetParent](#) method. To verify if an item can be parent for another item you can call [AcceptSetParent](#) property. If the item has no parent the ItemParent property retrieves 0. If the ItemParent gets 0 for an item, than the item is called root. The control is able to handle more root items. Use the [RootCount](#) and [RootItem](#) properties to enumerate the root items collection. Use the [InsertItem](#) method to insert a child item. The [NextVisibleItem](#) or [NextSiblingItem](#) gets the next visible, sibling item.

property Items.ItemPosition(Item as HITEM) as Long

Retrieves or sets a value that indicates the item's position in the children list.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Long	A long expression that indicates the item's position in the children list.

The ItemPosition property gets the item's position in the children items list. When the control sorts a column the position for each item is changed based on the sorting criteria. Use the item's handle to identify an item. Use the [ItemByIndex](#) property to get the handle of the item giving its index. Use the [ItemChild](#) property to get the first child item. Use the [NextVisibleItem](#) property to get the next visible item. The [NextSiblingItem](#) property retrieves the next sibling of the item in the parent's child list. Use the [SortChildren](#) method to sort the child items. Use the [SortOrder](#) property to sort a column.

property Items.ItemStrikeOut(Item as HITEM) as Boolean

Retrieves or sets the StrikeOut property of the Font object used to paint the item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item uses strikeout font attribute to paint 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 [CellCaptionFormat](#) 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 selected item:

```
Private Sub ComboBox1_SelectionChanged()  
    If Not (h = 0) Then ComboBox1.Items.ItemStrikeOut(h) = False  
    h = ComboBox1.Items.SelectedItem()  
    ComboBox1.Items.ItemStrikeOut(h) = True  
End Sub
```

The following VB sample draws a horizontal line through the focused item:

```
With ComboBox1.Items  
    .ItemStrikeOut(.FocusItem) = True  
End With
```

The following C++ sample draws a horizontal line through the focused item:

```
#include "Items.h"  
CItems items = m_combobox.GetItems();  
items.SetItemStrikeOut( items.GetFocusItem() , TRUE );
```

The following C# sample draws a horizontal line through the focused item:

```
axComboBox1.Items.set_ItemStrikeOut(axComboBox1.Items.FocusItem, true);
```

The following VB.NET sample draws a horizontal line through the focused item:

```
With AxComboBox1.Items  
    .ItemStrikeOut(.FocusItem) = True  
End With
```

The following VFP sample draws a horizontal line through the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .ItemStrikeOut( 0 ) = .t.  
endwith
```

property Items.ItemToIndex (Item as HITEM) as Long

Retrieves the index of an item in the Items collection, given its handle.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long expression that indicates the index of the item in Items collection.

Use the ItemToIndex property to get the item's index in the Items collection. Use [ItemPosition](#) property to change the item's position. Use the [ItemByIndex](#) property to get an item giving its index. The [ItemCount](#) property counts the items in the control. The [ChildCount](#) property counts the child items.

property Items.ItemUnderline(Item as HITEM) as Boolean

Retrieves or sets the Underline property of the Font object used to paint the item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates if the item is underlined or not. True is 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 [CellCaptionFormat](#) 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 selected item:

```
Private Sub ComboBox1_SelectionChanged()  
    If Not (h = 0) Then ComboBox1.Items.ItemUnderline(h) = False  
    h = ComboBox1.Items.SelectedItem()  
    ComboBox1.Items.ItemUnderline(h) = True  
End Sub
```

The following VB sample underlines the focused item:

```
With ComboBox1.Items  
    .ItemUnderline(FocusItem) = True  
End With
```

The following C++ sample underlines the focused item:

```
#include "Items.h"  
CItems items = m_combobox.GetItems();  
items.SetItemUnderline( items.GetFocusItem() , TRUE );
```

The following C# sample underlines the focused item:

```
axComboBox1.Items.set_ItemUnderline(axComboBox1.Items.FocusItem, true);
```

The following VB.NET sample underlines the focused item:

```
With AxComboBox1.Items  
    .ItemUnderline(.FocusItem) = True  
End With
```

The following VFP sample underlines the focused item:

```
with thisform.ComboBox1.Items  
    .DefaultItem = .FocusItem  
    .ItemUnderline( 0 ) = .t.  
endwith
```

property Items.LastVisibleItem as HITEM

Retrieves the handle of the last visible item.

Type	Description
HITEM	A long expression that indicates the handle of the last visible item.

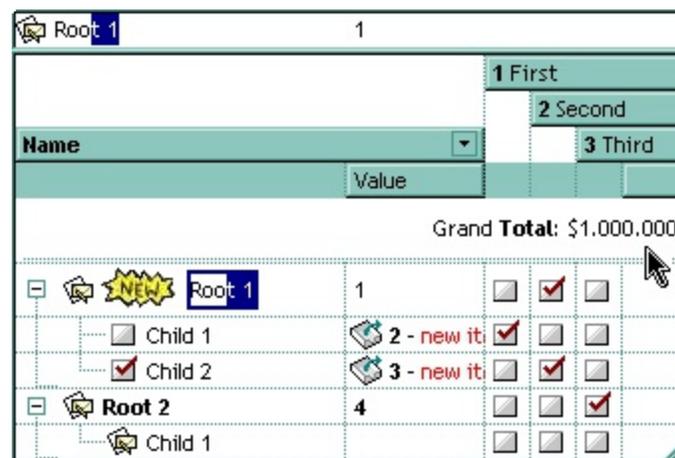
To get the first visible item use [FirstVisibleItem](#) property. The LastVisibleItem property retrieves the handle for the last visible item. Use the [FirstVisibleItem](#), [NextVisibleItem](#) and [IsItemVisible](#) properties to get the items that fit the client area. Use the [NextVisibleItem](#) property to get the next visible item. Use the [IsVisibleItem](#) property to check whether an item fits the control's client area.

property Items.LockedItem (Alignment as VAlignmentEnum, Index as Long) as HITEM

Retrieves the handle of the locked item. */*not supported in the lite version*/*

Type	Description
Alignment as VAlignmentEnum	A VAlignmentEnum expression that indicates whether the locked item requested is on the top or bottom side of the control.
Index as Long	A long expression that indicates the position of item being requested.
HITEM	A long expression that indicates the handle of the locked item.

A locked or fixed item is always displayed on the top or bottom side of the control no matter if the control's list is scrolled up or down. A locked item is not selectable. Use the LockedItem property to access a locked item by its position. Use the [LockedItemCount](#) property to add or remove items fixed/locked to the top or bottom side of the control. Use the [ShowLockedItems](#) property to show or hide the locked items. Use the [IsItemLocked](#) property to check whether an item is locked or unlocked. Use the [CellCaption](#) property to specify the caption for a cell. Use the [SelectableItem](#) property to indicate whether the item is selectable.



The following VB sample adds an item that's locked to the top side of the control:

With ComboBox1

```
Dim a As EXCOMBOBOXLibCtl.VAlignmentEnum  
a = EXCOMBOBOXLibCtl.VAlignmentEnum.exTop
```

```
.BeginUpdate
```

```
With .Items
```

```
.LockedItemCount(a) = 1
```

```

Dim h As EXCOMBOBOXLibCtl.HITEM
h = .LockedItem(a, 0)
.CellCaption(h, 0) = "<b>locked</b> item"
.CellCaptionFormat(h, 0) = exHTML
End With
.EndUpdate
End With

```

The following C++ sample adds an item that's locked to the top side of the control:

```

#include "Items.h"
m_combobox.BeginUpdate();
CItems items = m_combobox.GetItems();
items.SetLockedItemCount( 0 /*exTop*/, 1);
long i = items.GetLockedItem( 0 /*exTop*/, 0 );
COleVariant vtItem(i), vtColumn( long(0) );
items.SetCellCaption( vtItem, vtColumn, COleVariant( "<b>locked</b> item" ) );
items.SetCellCaptionFormat( vtItem, vtColumn, 1/*exHTML*/ );
m_combobox.EndUpdate();

```

The following VB.NET sample adds an item that's locked to the top side of the control:

```

With AxComboBox1
.BeginUpdate()
With .Items
.LockedItemCount(EXCOMBOBOXLib.VAlignmentEnum.exTop) = 1
Dim i As Integer
i = .LockedItem(EXCOMBOBOXLib.VAlignmentEnum.exTop, 0)
.CellCaption(i, 0) = "<b>locked</b> item"
.CellCaptionFormat(i, 0) = EXCOMBOBOXLib.CaptionFormatEnum.exHTML
End With
.EndUpdate()
End With

```

The following C# sample adds an item that's locked to the top side of the control:

```

axComboBox1.BeginUpdate();
EXCOMBOBOXLib.Items items = axComboBox1.Items;
items.set_LockedItemCount(EXCOMBOBOXLib.VAlignmentEnum.exTop, 1);

```

```
int i = items.get_LockedItem(EXCOMBOBOXLib.VAlignmentEnum.exTop, 0);
items.set_CellCaption(i, 0, "<b>locked</b> item");
items.set_CellCaptionFormat(i, 0, EXCOMBOBOXLib.CaptionFormatEnum.exHTML);
axComboBox1.EndUpdate();
```

The following VFP sample adds an item that's locked to the top side of the control:

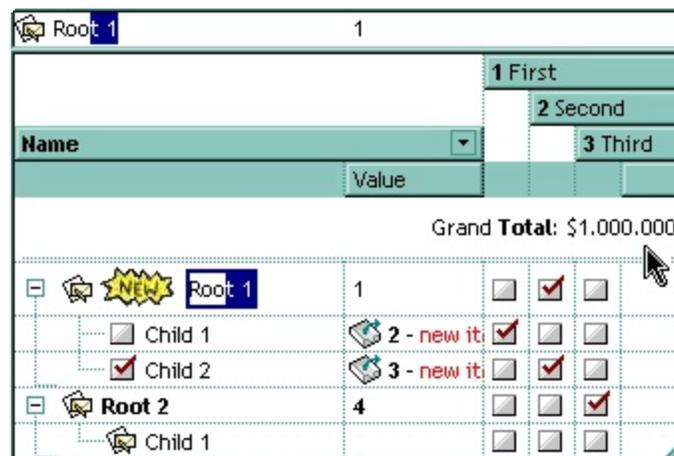
```
with thisform.ComboBox1
  .BeginUpdate()
  With .Items
    .LockedItemCount(0) = 1
    .DefaultItem = .LockedItem(0, 0)
    .CellCaption(0, 0) = "<b>locked</b> item"
    .CellCaptionFormat(0, 0) = 1 && EXCOMBOBOXLib.CaptionFormatEnum.exHTML
  EndWith
  .EndUpdate()
endwith
```

property Items.LockedItemCount(Alignment as VAlignmentEnum) as Long

Specifies the number of items fixed on the top or bottom side of the control. */*not supported in the lite version*/*

Type	Description
Alignment as VAlignmentEnum	A VAlignmentEnum expression that specifies the top or bottom side of the control.
Long	A long expression that indicates the number of items locked to the top or bottom side of the control.

A locked or fixed item is always displayed on the top or bottom side of the control no matter if the control's list is scrolled up or down. A locked item is not selectable. Use the LockedItemCount property to add or remove items fixed/locked to the top or bottom side of the control. Use the [LockedItem](#) property to access a locked item by its position. Use the [ShowLockedItems](#) property to show or hide the locked items. Use the [CellCaption](#) property to specify the caption for a cell. Use the [CountLockedColumns](#) property to lock or unlock columns in the control. Use the [ItemBackColor](#) property to specify the item's background color. Use the [ItemDivider](#) property to merge the cells. Use the [MergeCells](#) method to combine two or more cells in a single cell. Use the [SelectableItem](#) property to indicate whether the item is selectable.



The following VB sample adds an item that's locked to the top side of the control:

With ComboBox1

```
Dim a As EXCOMBOBOXLibCtl.VAlignmentEnum
a = EXCOMBOBOXLibCtl.VAlignmentEnum.exTop
.BeginUpdate
With .Items
.LockedItemCount(a) = 1
```

```

Dim h As EXCOMBOBOXLibCtl.HITEM
h = .LockedItem(a, 0)
.CellCaption(h, 0) = "<b>locked</b> item"
.CellCaptionFormat(h, 0) = exHTML
End With
.EndUpdate
End With

```

The following C++ sample adds an item that's locked to the top side of the control:

```

#include "Items.h"
m_combobox.BeginUpdate();
CItems items = m_combobox.GetItems();
items.SetLockedItemCount( 0 /*exTop*/, 1);
long i = items.GetLockedItem( 0 /*exTop*/, 0 );
COleVariant vtItem(i), vtColumn( long(0) );
items.SetCellCaption( vtItem, vtColumn, COleVariant( "<b>locked</b> item" ) );
items.SetCellCaptionFormat( vtItem, vtColumn, 1/*exHTML*/ );
m_combobox.EndUpdate();

```

The following VB.NET sample adds an item that's locked to the top side of the control:

```

With AxComboBox1
.BeginUpdate()
With .Items
.LockedItemCount(EXCOMBOBOXLib.VAlignmentEnum.exTop) = 1
Dim i As Integer
i = .LockedItem(EXCOMBOBOXLib.VAlignmentEnum.exTop, 0)
.CellCaption(i, 0) = "<b>locked</b> item"
.CellCaptionFormat(i, 0) = EXCOMBOBOXLib.CaptionFormatEnum.exHTML
End With
.EndUpdate()
End With

```

The following C# sample adds an item that's locked to the top side of the control:

```

axComboBox1.BeginUpdate();
EXCOMBOBOXLib.Items items = axComboBox1.Items;
items.set_LockedItemCount(EXCOMBOBOXLib.VAlignmentEnum.exTop, 1);

```

```
int i = items.get_LockedItem(EXCOMBOBOXLib.VAlignmentEnum.exTop, 0);
items.set_CellCaption(i, 0, "<b>locked</b> item");
items.set_CellCaptionFormat(i, 0, EXCOMBOBOXLib.CaptionFormatEnum.exHTML);
axComboBox1.EndUpdate();
```

The following VFP sample adds an item that's locked to the top side of the control:

```
with thisform.ComboBox1
  .BeginUpdate()
  With .Items
    .LockedItemCount(0) = 1
    .DefaultItem = .LockedItem(0, 0)
    .CellCaption(0, 0) = "<b>locked</b> item"
    .CellCaptionFormat(0, 0) = 1 && EXCOMBOBOXLib.CaptionFormatEnum.exHTML
  EndWith
  .EndUpdate()
endwith
```

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 ([ItemCount](#) property). If negative, a filter is applied, and the absolute value minus one, indicates the number of matching items after filter is applied. A matching item includes its parent items, if the control's [FilterInclude](#) property allows including child items.

The MatchItemCount property returns a value as explained bellow:

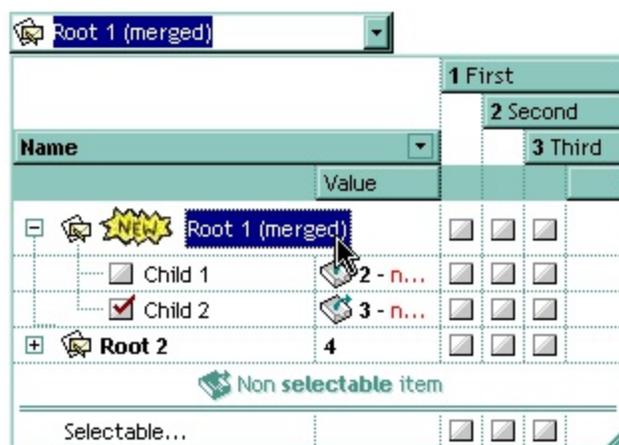
- 0, the control displays/contains no items, and no filter is applied to any column
- -1, the control displays no items, and there is a filter applied (no match found)
- positive number, indicates the number of items within the control ([ItemCount](#) property)
- negative number, the absolute value minus 1, indicates the number of items that matches the current filter (match found)

method Items.MergeCells ([Cell1 as Variant], [Cell2 as Variant], [Options as Variant])

Merges a list of cells. */*not supported in the lite version*/*

Type	Description
Cell1 as Variant	A long expression that indicates the handle of the cell being merged, or a safe array that holds a collection of handles for the cells being merged. Use the ItemCell property to retrieve the handle of the cell. The first cell (in the list, if exists) specifies the cell being displayed in the new larger cell.
Cell2 as Variant	A long expression that indicates the handle of the cell being merged, or a safe array that holds a collection of handles for the cells being merged. Use the ItemCell property to retrieve the handle of the cell. The first cell in the list specifies the cell being displayed in the new larger cell.
Options as Variant	Reserved.

The MergeCells method combines two or more cells into one cell. The data in the **first specified cell** is displayed in the new larger cell. All the other cells' data is not lost. Use the [CellMerge](#) property to merge or unmerge a cell with another cell in the same item. Use the [ItemDivider](#) property to display a single cell in the entire item. Use the [UnmergeCells](#) method to unmerge the merged cells. Use the [CellCaption](#) property to specify the cell's caption. Use the [ItemCell](#) property to retrieve the handle of the cell. Use the [BeginMethod](#) and [EndUpdate](#) methods to maintain performance, when merging multiple cells in the same time. The MergeCells methods creates a list of cells from Cell1 and Cell2 parameters that need to be merged, and the first cell in the list specifies the displayed cell in the merged cell. Use the [LockedItemCount](#) property to add or remove items fixed/locked to the top or bottom side of the control.



The following samples adds three columns, a root item and two child items:

With Tree1

```
.BeginUpdate
  .MarkSearchColumn = False
  .DrawGridLines = exAllLines
  .LinesAtRoot = exLinesAtRoot
  With .Columns.Add("Column 1")
    .Def(exCellCaptionFormat) = exHTML
  End With
  .Columns.Add "Column 2"
  .Columns.Add "Column 3"
  With .Items
    Dim h As Long
    h = .AddItem("Root. This is the root item")
    .InsertItem h, , Array("Child 1", "SubItem 2", "SubItem 3")
    .InsertItem h, , Array("Child 2", "SubItem 2", "SubItem 3")
    .ExpandItem(h) = True
    .SelectItem(h) = True
  End With
.EndUpdate
End With
```

and it looks like follows (notice that the caption of the root item is truncated by the column that belongs to):

Column 1	Column 2	Column 3
<input type="checkbox"/> Root. This is		
<input type="checkbox"/> Child 1	SubItem 2	SubItem 3
<input type="checkbox"/> Child 2	SubItem 2	SubItem 3

If we are merging the first three cells in the root item we get:

Column 1	Column 2	Column 3
<input type="checkbox"/> Root. This is the root item		
<input type="checkbox"/> Child 1	SubItem 2	SubItem 3
<input type="checkbox"/> Child 2	SubItem 2	SubItem 3

You can merge the first three cells in the root item using any of the following methods:

With ComboBox1

With .Items

```
.CellMerge(.RootItem(0), 0) = Array(1, 2)
```

```
End With
```

```
End With
```

```
With ComboBox1
```

```
.BeginUpdate
```

```
With .Items
```

```
Dim r As Long
```

```
r = .RootItem(0)
```

```
.CellMerge(r, 0) = 1
```

```
.CellMerge(r, 0) = 2
```

```
End With
```

```
.EndUpdate
```

```
End With
```

```
With ComboBox1
```

```
.BeginUpdate
```

```
With .Items
```

```
Dim r As Long
```

```
r = .RootItem(0)
```

```
.MergeCells .ItemCell(r, 0), .ItemCell(r, 1)
```

```
.MergeCells .ItemCell(r, 0), .ItemCell(r, 2)
```

```
End With
```

```
.EndUpdate
```

```
End With
```

```
With ComboBox1
```

```
With .Items
```

```
Dim r As Long
```

```
r = .RootItem(0)
```

```
.MergeCells .ItemCell(r, 0), Array(.ItemCell(r, 1), .ItemCell(r, 2))
```

```
End With
```

```
End With
```

```
With ComboBox1
```

```
With .Items
```

```
Dim r As Long
```

```
r = .RootItem(0)
.MergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1), .ItemCell(r, 2))
End With
End With
```

The following VB sample merges the first three cells:

```
With ComboBox1.Items
.MergeCells .ItemCell(.FocusItem, 0), Array(.ItemCell(.FocusItem, 1), .ItemCell(.FocusItem,
2))
End With
```

The following C++ sample merges the first three cells:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
COleVariant vtFocusCell( items.GetItemCell(items.GetFocusItem(), COleVariant( (long)0 ) ) ),
vtMissing; V_VT( &vtMissing ) = VT_ERROR;
items.MergeCells( vtFocusCell, COleVariant( items.GetItemCell(items.GetFocusItem(),
COleVariant( (long)1 ) ) ), vtMissing );
items.MergeCells( vtFocusCell, COleVariant( items.GetItemCell(items.GetFocusItem(),
COleVariant( (long)2 ) ) ), vtMissing );
```

The following VB.NET sample merges the first three cells:

```
With AxComboBox1.Items
.MergeCells(.ItemCell(.FocusItem, 0), .ItemCell(.FocusItem, 1))
.MergeCells(.ItemCell(.FocusItem, 0), .ItemCell(.FocusItem, 2))
End With
```

The following C# sample merges the first three cells:

```
EXCOMBOBOXLib.Items items = axComboBox1.Items;
items.MergeCells(items.get_ItemCell( items.FocusItem, 0 ), items.get_ItemCell(
items.FocusItem, 1 ), "");
items.MergeCells(items.get_ItemCell(items.FocusItem, 0),
items.get_ItemCell(items.FocusItem, 2), "");
```

The following VFP sample merges the first three cells:

```
with thisform.ComboBox1.Items
```

```
    .MergeCells(.ItemCell(.FocusItem,0), .ItemCell(.FocusItem,1), "")
```

```
    .MergeCells(.ItemCell(.FocusItem,0), .ItemCell(.FocusItem,2), "")
```

```
endwith
```

Now, the question is what should I use in my program in order to merge some cells? For instance, if you are using handle to cells (HCELL type), we would recommend using the MergeCells method, else you could use as well the CellMerge property

property Items.NextSiblingItem (Item as HITEM) as HITEM

Retrieves the next sibling of the item in the parent's child list.

Type	Description
Item as HITEM	A long expression that indicates the item's handle
HITEM	A long expression that indicates the handle of the next sibling item.

The NextSiblingItem property retrieves the next sibling of the item in the parent's child list. Use [ItemChild](#) and NextSiblingItem properties to enumerate the collection of child items. Use the [ItemParent](#) property to retrieve the handle of the parent item.

The following VB function recursively enumerates the item and all its child items:

```
Sub Recltem(ByVal c As EXCOMBOBOXLibCtl.ComboBox, ByVal h As HITEM)
    If Not (h = 0) Then
        Dim hChild As HITEM
        With c.Items
            Debug.Print .CellCaption(h, 0)
            hChild = .ItemChild(h)
            While Not (hChild = 0)
                Recltem c, hChild
                hChild = .NextSiblingItem(hChild)
            Wend
        End With
    End If
End Sub
```

The following C++ function recursively enumerates the item and all its child items:

```
void Recltem( CComboBox* pComboBox, long hItem )
{
    COleVariant vtColumn( (long)0 );
    if ( hItem )
    {
        CItems items = pComboBox->GetItems();

        CString strCaption = V2S( &items.GetCellCaption( COleVariant( hItem ), vtColumn ) ),
```

```

strOutput;
    strOutput.Format( "Cell: '%s'\n", strCaption );
    OutputDebugString( strOutput );

    long hChild = items.GetItemChild( hItem );
    while ( hChild )
    {
        Recltem( pComboBox, hChild );
        hChild = items.GetNextSiblingItem( hChild );
    }
}
}
}

```

The following VB.NET function recursively enumerates the item and all its child items:

```

Shared Sub Recltem(ByVal c As AxEXCOMBOBOXLib.AxComboBox, ByVal h As Integer)
    If Not (h = 0) Then
        Dim hChild As Integer
        With c.Items
            Debug.WriteLine(.CellCaption(h, 0))
            hChild = .ItemChild(h)
            While Not (hChild = 0)
                Recltem(c, hChild)
                hChild = .NextSiblingItem(hChild)
            End While
        End With
    End If
End Sub

```

The following C# function recursively enumerates the item and all its child items:

```

internal void Recltem(AxEXCOMBOBOXLib.AxComboBox combobox, int hItem)
{
    if (hItem != 0)
    {
        EXCOMBOBOXLib.Items items = combobox.Items;
        object caption = items.get_CellCaption( hItem, 0 );
        System.Diagnostics.Debug.WriteLine(caption != null ? caption.ToString() : "");
    }
}

```

```

int hChild = items.get_ItemChild(hItem);
while (hChild != 0)
{
    Recltem(combobox, hChild);
    hChild = items.get_NextSiblingItem(hChild);
}
}
}

```

The following VFP function recursively enumerates the item and all its child items (recitem method):

```

LPARAMETERS h

with thisform.ComboBox1
  If ( h != 0 ) Then
    local hChild
    With .Items
      .DefaultItem = h
      wait window .CellCaption(0, 0)
      hChild = .ItemChild(h)
      do While (hChild != 0)
        thisform.recitem(hChild)
        hChild = .NextSiblingItem(hChild)
      enddo
    EndWith
  EndIf
endwith

```

property Items.NextVisibleItem (Item as HITEM) as HITEM

Retrieves the handle of next visible item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the handle of the next visible item.

Use the NextVisibleItem property to access the visible items. The NextVisibleItem property retrieves 0 if there are no more visible items. Use the [IsItemVisible](#) property to check whether an item fits the control's client area. Use the [ItemChild](#) property to get the first child item. Use the [ItemParent](#) property to retrieve the handle of the parent item.

The following VB sample enumerates all visible items in the control's list:

```
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = ComboBox1.Columns.Count
With ComboBox1.Items
    h = .FirstVisibleItem
    While Not (h = 0)
        Dim s As String
        s = ""
        For j = 0 To nCols - 1
            s = s + .CellCaption(h, j) + Chr(9)
        Next
        Debug.Print s
        h = .NextVisibleItem(h)
    Wend
End With
```

The following C++ sample enumerates all visible items in the control's list:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
long hItem = items.GetFirstVisibleItem();
while ( hItem && items.GetIsItemVisible( hItem ) )
{
```

```

OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hltem ), COleVariant(
long(0) ) ) ) );
hltem = items.GetNextVisibleItem( hltem );
}

```

The following VB.NET sample enumerates all visible items in the control's list:

```

With AxComboBox1.Items
Dim hltem As Integer
hltem = .FirstVisibleItem
While Not (hltem = 0)
Debug.Print(.CellCaption(hltem, 0))
hltem = .NextVisibleItem(hltem)
End While
End With

```

The following C# sample enumerates all visible items in the control's list:

```

EXCOMBOBOXLib.Items items = axComboBox1.Items;
int hltem = items.FirstVisibleItem;
while (hltem != 0)
{
object strCaption = items.get_CellCaption(hltem, 0);
System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
hltem = items.get_NextVisibleItem(hltem);
}

```

The following VFP sample all visible items in the control's list:

```

with thisform.ComboBox1.Items
.DefaultItem = .FirstVisibleItem
do while ( .DefaultItem <> 0 )
wait window .CellCaption( 0, 0 )
.DefaultItem = .NextVisibleItem( 0 )
enddo
endwith

```

property Items.PathSeparator as String

Returns or sets the delimiter character used for the path returned by the [FullPath](#) and [FindPath](#) properties.

Type	Description
String	A string expression that indicates the delimiter character used for the path returned by the FullPath and FindPath properties.

By default the PathSeparator is "\". The PathSeparator property is used by properties like [FullPath](#) and [FindPath](#).

property Items.PrevSiblingItem (Item as HITEM) as HITEM

Retrieves the previous sibling of the item in the parent's child list.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the handle of the previous sibling item.

The `PrevSiblingItem` retrieves 0 if there are no more previous sibling items. The [NextSiblingItem](#) property retrieves the next sibling of the item in the parent's child list. Use the [FirstVisibleItem](#) property to retrieve the first visible item. Use the [ItemParent](#) property to retrieve the parent of the item.

property Items.PrevVisibleItem (Item as HITEM) as HITEM

Retrieves the handle of previous visible item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the handle of the previous visible item.

The `PrevVisibleItem` property retrieves 0 if there are no previous visible items. The [NextVisibleItem](#) property retrieves the next visible item. Use the [FirstVisibleItem](#) property to retrieve the first visible item. Use the [ItemParent](#) property to retrieve the parent of the item.

method `Items.RemoveAllItems ()`

Removes all items from the control.

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

`RemoveAllItems` clears the `Items` collection. The [Clear](#) method clears the columns collection and remove all items too. Use the [RemoveItem](#) method to remove an item from the `Items` collection. The control fires the [RemoveItem](#) event when the user deletes an item.

method Items.RemoveItem (Item as HITEM)

Removes the given item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle being removed.

The RemoveItem method removes an item. The RemoveItem method does not remove the item, if it contains child items. The [RemoveSelection](#) method removes the selected items (including the descendents). The following sample removes the first item: `ComboBox1.Items.RemoveItem ComboBox1.Items(0)`. Use the [RemoveAllItems](#) method to remove all items in the control. Use the [BeginUpdate](#) and [EndUpdate](#) methods to maintain performance while removing the items. The RemoveItem method can't remove an item that's locked. Instead you can use the [LockedItemCount](#) property to add or remove locked items. Use the [IsItemLocked](#) property to check whether an item is locked. If the [Style](#) property is `DropDown`, you can use the [EditText](#) property to specify the text in the control's label area after removing an item.

The following VB sample removes recursively an item:

```
Private Sub RemoveItemRec(ByVal t As EXCOMBOBOXLibCtl.ComboBox, ByVal h As HITEM)
    If Not h = 0 Then
        With t.Items
            t.BeginUpdate
            Dim hChild As HITEM
            hChild = .ItemChild(h)
            While (hChild <> 0)
                Dim hNext As HITEM
                hNext = .NextSiblingItem(hChild)
                RemoveItemRec t, hChild
                hChild = hNext
            Wend
            .RemoveItem h
            t.EndUpdate
        End With
    End If
End Sub
```

The following C++ sample removes recursively an item:

```
void RemoveItemRec( CComboBox* pComboBox, long hItem )
{
    if ( hItem )
    {
        pComboBox->BeginUpdate();
        CItems items = pComboBox->GetItems();
        long hChild = items.GetItemChild( hItem );
        while ( hChild )
        {
            long nNext = items.GetNextSiblingItem( hChild );
            RemoveItemRec( pComboBox, hChild );
            hChild = nNext;
        }
        items.RemoveItem( hItem );
        pComboBox->EndUpdate();
    }
}
```

The following VB.NET sample removes recursively an item:

```
Shared Sub RemoveItemRec(ByVal t As AxEXCOMBOBOXLib.AxComboBox, ByVal h As Integer)
    If Not h = 0 Then
        With t.Items
            t.BeginUpdate()
            Dim hChild As Integer = .ItemChild(h)
            While (hChild <> 0)
                Dim hNext As Integer = .NextSiblingItem(hChild)
                RemoveItemRec(t, hChild)
                hChild = hNext
            End While
            .RemoveItem(h)
            t.EndUpdate()
        End With
    End If
End Sub
```

The following C# sample removes recursively an item:

```
internal void RemoveItemRec(AxEXCOMBOBOXLib.AxComboBox combobox, int hItem)
{
    if (hItem != 0)
    {
        EXCOMBOBOXLib.Items items = combobox.Items;
        combobox.BeginUpdate();
        int hChild = items.get_ItemChild(hItem);
        while (hChild != 0)
        {
            int hNext = items.get_NextSiblingItem(hChild);
            RemoveItemRec(combobox, hChild);
            hChild = hNext;
        }
        items.RemoveItem(hItem);
        combobox.EndUpdate();
    }
}
```

The following VFP sample removes recursively an item (removeitemrec method):

LPARAMETERS h

with thisform.ComboBox1

 If (h != 0) Then

 .BeginUpdate()

 local hChild

 With .Items

 hChild = .ItemChild(h)

 do While (hChild != 0)

 local hNext

 hNext = .NextSiblingItem(hChild)

 thisform.removeitemrec(hChild)

 hChild = hNext

 enddo

 .RemoveItem(h)

 EndWith

```
.EndUpdate()
```

```
EndIf
```

```
endwith
```

method `Items.RemoveSelection ()`

Removes the selected items (including the descendents).

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

The `RemoveSelection` method removes the selected items (including the descendents). The [RemoveItem](#) method removes a specific item (if it has no child items). The [UnselectAll](#) method unselects all items.

property Items.RootCount as Long

Retrieves the number of root objects in the Items collection.

Type	Description
Long	A long value that indicates the count of root items into Items collection.

A root item is an item that has no parent ([ItemParent\(\)](#) = 0). Use the [RootItem](#) property of the Items object to enumerates the root items. Use the [AddItem](#) to add root items to the control. Use the [InsertItem](#) method to insert child items.

The following VB sample enumerates all root items:

```
Dim i As Long, n As Long
With ComboBox1.Items
    n = .RootCount
    For i = 0 To n - 1
        Debug.Print .CellCaption(.RootItem(i), 0)
    Next
End With
```

The following C++ sample enumerates all root items:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
for ( long i = 0 ; i < items.GetRootCount(); i++ )
{
    COleVariant vtItem( items.GetRootItem(i) ), vtColumn( long(0) );
    OutputDebugString( V2S( &items.GetCellCaption( vtItem, vtColumn ) ) );
}
```

The following VB.NET sample enumerates all root items:

```
With AxComboBox1.Items
    Dim i As Integer
    For i = 0 To .RootCount - 1
        Debug.Print(.CellCaption(.RootItem(i), 0))
    Next
End With
```

The following C# sample enumerates all root items:

```
for (int i = 0; i < axComboBox1.Items.RootCount; i++)
{
    object strCaption =
axComboBox1.Items.get_CellCaption(axComboBox1.Items.get_RootItem(i), 0);
    System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following VFP sample enumerates all root items:

```
with thisform.ComboBox1.Items
    local i
    for i = 0 to .RootCount - 1
        .DefaultItem = .RootItem(i)
        wait window nowait .CellCaption(0,0)
    next
endwith
```

property Items.RootItem ([Position as Long]) as HITEM

Retrieves the handle of the root item given its index in the root items collection.

Type	Description
Position as Long	A long value that indicates the position of the root item.
HITEM	A long expression that specifies the handle of the the root item.

A root item is an item that has no parent ([ItemParent\(\)](#) = 0). Use the [RootCount](#) property of to count the root items. Use the [AddItem](#) to add root items to the control. Use the [InsertItem](#) method to insert child items.

The following VB sample enumerates all root items:

```
Dim i As Long, n As Long
With ComboBox1.Items
    n = .RootCount
    For i = 0 To n - 1
        Debug.Print .CellCaption(.RootItem(i), 0)
    Next
End With
```

The following C++ sample enumerates all root items:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
for ( long i = 0 ; i < items.GetRootCount(); i++ )
{
    COleVariant vtItem( items.GetRootItem(i) ), vtColumn( long(0) );
    OutputDebugString( V2S( &items.GetCellCaption( vtItem, vtColumn ) ) );
}
```

The following VB.NET sample enumerates all root items:

```
With AxComboBox1.Items
    Dim i As Integer
    For i = 0 To .RootCount - 1
        Debug.Print(.CellCaption(.RootItem(i), 0))
    Next
```

The following C# sample enumerates all root items:

```
for (int i = 0; i < axComboBox1.Items.RootCount; i++)
{
    object strCaption =
axComboBox1.Items.get_CellCaption(axComboBox1.Items.get_RootItem(i), 0);
    System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following VFP sample enumerates all root items:

```
with thisform.ComboBox1.Items
    local i
    for i = 0 to .RootCount - 1
        .DefaultItem = .RootItem(i)
        wait window nowait .CellCaption(0,0)
    next
endwith
```

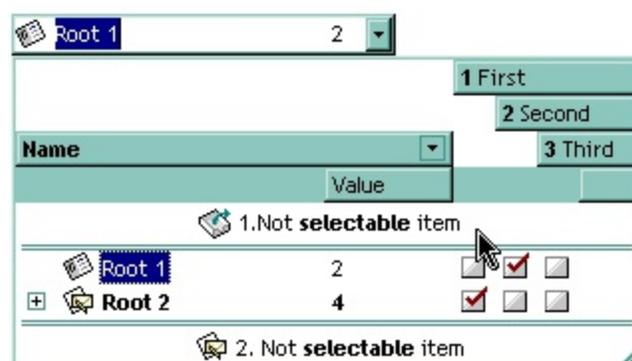
property Items.SelectableItem(Item as HITEM) as Boolean

Specifies whether the user can select the item. */*not supported in the lite version*/*

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being selectable
Boolean	A boolean expression that specifies whether the item is selectable.

By default, all items are selectable, excepts the locked items that are not 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. The [LockedItemCount](#) property specifies the number of locked items to the top or bottom side of the control. Use the [ItemForeColor](#) property to specifies 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](#) or [Select](#) 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](#) or [Change](#) 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. If the user selects an item, and then he declares the item as non selectable after, the control clears the control's label and its selection. Use the [SelectCount](#) property to get the number of selected items. The control highlights a selectable item when cursor hovers the item, and doesn't highlight a non selectable item. Use the [SelfForeColor](#) and [SelBackColor](#) properties to customize the colors for selected items.

For instance, if the user clicks a non selectable item, the control doesn't close its drop down portion, so it waits until the user selects a new item.



The following VB sample makes not selectable the first visible item:

```
With ComboBox1.Items
    .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C++ sample makes not selectable the first visible item:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
items.SetSelectableItem( items.GetFirstVisibleItem(), FALSE );
```

The following VB.NET sample makes not selectable the first visible item:

```
With AxComboBox1.Items
    .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C# sample makes not selectable the first visible item:

```
axComboBox1.Items.set_SelectableItem(axComboBox1.Items.FirstVisibleItem, false);
```

The following VFP sample makes not selectable the first visible item:

```
with thisform.ComboBox1.Items
    .DefaultItem = .FirstVisibleItem
    .SelectableItem(0) = .f.
endwith
```

method Items.SelectAll ()

Selects all items.

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

Use the `SelectAll`, [UnselectAll](#) method to select / unselect all items within the control (mostly when the [SingleSel](#) property is `False`). The [SelectCount](#) property counts the selected items in the control. Use the [SelectedItem](#) property to retrieve the handle of the selected item. Use the [Value](#) property to select a value in a single column control. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectItem](#) or [Select](#) property to select an item.

property Items.SelectCount as Long

Counts the number of items that are selected in the control.

Type	Description
Long	A long expression that identifies the number of selected items.

The SelectCount property counts the selected items in the control. The [SingleSel](#) property retrieves or sets a value that indicates whether the control supports single or multiple selection. If the control contains a selected item, the SelectCount property is 1. If there is no selected items, the SelectCount property is 0. Use the [SelectedItem](#) property to retrieve the handle of the selected item. Use the [Value](#) property to select a value in a single column control. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectItem](#) or [Select](#) property to select an item.

property Items.SelectedItem ([Index as Long]) as HITEM

Retrieves the selected item's handle given its index in selected items collection.

Type	Description
Index as Long	Identifies the index of the selected item into selected items collection.
HITEM	A long expression that indicates the handle of the selected item.

Currently, The control supports only single selection. Use the [SelectCount](#) property to determine whether the control has an item selected. Use the [CellCaption](#) property to get the caption of a specified cell. Use the [Value](#) property to get the selected value on a single column control. If the control contains multiple columns you can use the [Select](#) property to get the selected value on a specified column.

The following VB sample displays the selected item, using the SelectedItem property:

```
With ComboBox1.Items
  Dim h As HITEM
  h = .SelectedItem()
  If (Not h = 0) Then
    Debug.Print .CellCaption(h, 0)
  End If
End With
```

The following C++ sample displays the selected item, using the SelectedItem property:

```
CItems items = m_combobox.GetItems();
long hItem = items.GetSelectedItem( 0 );
if ( hItem != 0 )
  OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
```

where the V2S function converts a VARIANT value to a string,

```
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 the selected item, using the SelectedItem property:

```

With AxComboBox1.Items
    Dim h As Integer = .SelectedItem()
    If (Not h = 0) Then
        Debug.WriteLine(.CellCaption(h, 0))
    End If
End With

```

The following C# sample displays the selected item, using the SelectedItem property:

```

EXCOMBOBOXLib.Items items = axComboBox1.Items;
int hItem = items.get_SelectedItem(0);
if ( hItem != 0 )
    System.Diagnostics.Debug.WriteLine(items.get_CellCaption(hItem,0).ToString());

```

The following VFP sample displays the selected item, using the SelectedItem property:

```

With thisform.ComboBox1.Items
    .DefaultItem = .SelectedItem(0)
    If (.DefaultItem <> 0) Then
        wait window nowait .CellCaption(0, 0)
    EndIf
EndWith

```

property Items.SelectItem(Item as HITEM) as Boolean

Selects or unselects a specific item.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being selected or un-selected.
Boolean	A boolean expression that indicates the item's state. True if the item is selected, and False if the item is not selected.

The SelectItem property selects an item. The control selects only selectable items. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectedItem](#) property to retrieve the selected item. Use the [Select](#) property to select an item given a value on a specified column. Use the [Value](#) property to select an item given its value on a single column control. Use the [ItemByIndex](#) property to access an item given its index. The control fires the [SelectionChanged](#) event when user selects an item. Use the [SelForeColor](#) and [SelBackColor](#) properties to specify colors for selected items.

The following VB sample selects the first visible item:

```
With ComboBox1.Items
    .SelectItem(.FirstVisibleItem) = True
End With
```

The following C++ sample selects the first visible item:

```
#include "Items.h"
CItems items = m_combobox.GetItems();
items.SetSelectItem( items.GetFirstVisibleItem(), TRUE );
```

The following VB.NET sample selects the first visible item:

```
With AxComboBox1.Items
    .SelectItem(.FirstVisibleItem) = True
End With
```

The following C# sample selects the first visible item:

```
axComboBox1.Items.set_SelectItem(axComboBox1.Items.FirstVisibleItem, true);
```

The following VFP sample selects the first visible item:

```
with thisform.ComboBox1.Items  
  .DefaultItem = .FirstVisibleItem  
  .SelectedItem(0) = .t.  
endwith
```

property Items.SelectPos as Variant

Selects items by position.

Type	Description
Variant	A long expression that indicates the position of item being selected, or a safe array that holds a collection of position of items being selected.

Use the SelectPos property to select items by position. Use the [SelectItem](#) property to select an item giving its handle. The SelectPos property selects an item giving its general position. The [ItemPosition](#) property gives the relative position, or the position of the item in the child items collection.

The following VB sample selects the first item in the control:

```
Tree1.Items.SelectPos = 0
```

The following VB sample selects first two items:

```
Tree1.Items.SelectPos = Array(0, 1)
```

The following C++ sample selects the first item in the control:

```
m_tree.GetItems().SetSelectPos( COleVariant( long(0) ) );
```

The following VB.NET sample selects the first item in the control:

```
With AxTree1.Items  
    .SelectPos = 0  
End With
```

The following C# sample selects the first item in the control:

```
axTree1.Items.SelectPos = 0;
```

The following VFP sample selects the first item in the control:

```
with thisform.Tree1.Items  
    .SelectPos = 0  
endwith
```


method `Items.SetParent` (Item as HITEM, NewParent as HITEM)

Changes the parent of the given item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
NewParent as HITEM	A long expression that indicates the handle of the newly parent item.

Use the `SetParent` property to change the parent item at runtime. Use the [InsertItem](#) property to insert child items. Use [AcceptSetParent](#) property to verify if the the parent of an item can be changed. The following VB sample changes the parent item of the first item: `ComboBox1.Items.SetParent ComboBox1.Items(0), ComboBox1.Items(1)`. Use the [ItemParent](#) property to retrieve the parent of the item.

property Items.SortableItem(Item as HITEM) as Boolean

Specifies whether the item is sortable.

Type	Description
Item as HITEM	A long expression that indicates the handle 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. Though, 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 [SortChildren](#) method to sort the items. Use the [AllowSort](#) property to avoid sorting a column when the user clicks the column. The [ItemDivider](#) property indicates whether the item displays a single cell, instead showing all cells. 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)

Name	A	B	C
Group 1			
Child 1	1	2	3
Child 2	4	5	6
Group 2			
Child 1	1	2	3
Child 2	4	5	6

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)

Name	A	B	C
Group 1			
Child 2	4	5	6
Child 1	1	2	3
Group 2			
Child 2	4	5	6
Child 1	1	2	3

method `Items.SortChildren` (Item as `HITEM`, ColIndex as `Variant`, Ascending as `Boolean`)

Sorts the child items of the given parent item in the control.

Type	Description
Item as <code>HITEM</code>	A long expression that indicates the item's handle.
ColIndex as <code>Variant</code>	A long expression that indicates the column's index, a string expression that indicates the column's caption.
Ascending as <code>Boolean</code>	A boolean expression that defines the sort order.

`SortChildren` will not recurse through the `ComboBox`, only the immediate children of item will be sorted. If your control acts like a simple list you can use the following line of code to sort ascending the list by first column: `ComboBox1.Items.SortChildren 0, 0`. To change the way how a column is sorted use [SortType](#) property of `Column` object. The `SortChildren` property doesn't display the sort icon on column's header. The control automatically sorts the children items when user clicks on column's header. Use the [SortOnClick](#) property to disable sorting columns by clicking in the columns header. Use the [SortOrder](#) property to get the column sorted and to display the sorting icon in the column's header. Use the [AllowSort](#) property to avoid sorting a column when the user clicks the column.

method Items.UnmergeCells ([Cell as Variant])

Unmerges a list of cells. */*not supported in the lite version*/*

Type	Description
Cell as Variant	A long expression that indicates the handle of the cell being unmerged, or a safe array that holds a collection of handles for the cells being unmerged. Use the ItemCell property to retrieves the handle of the cell.

Use the UnmergeCells method to unmerge merged cells. Use the [MergeCells](#) method or [CellMerge](#) property to combine (merge) two or more cells in a single one. The UnmergeCells method unmerges all the cells that was merged. The CellMerge property unmerges only a single cell. The rest of merged cells remains combined.

The following sample shows few methods to unmerge cells:

```
With ComboBox1
  With .Items
    .UnmergeCells .ItemCell(.RootItem(0), 0)
  End With
End With
```

```
With ComboBox1
  With .Items
    Dim r As Long
    r = .RootItem(0)
    .UnmergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1))
  End With
End With
```

```
With ComboBox1
  .BeginUpdate
  With .Items
    .CellMerge(.RootItem(0), 0) = -1
    .CellMerge(.RootItem(0), 1) = -1
    .CellMerge(.RootItem(0), 2) = -1
  End With
  .EndUpdate
End With
```


method Items.UnselectAll ()

Unselects all items.

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

Use the [SelectAll](#), UnselectAll method to select / unselect all items within the control (mostly when the [SingleSel](#) property is False). The [SelectCount](#) property counts the selected items in the control. Use the [SelectedItem](#) property to retrieve the handle of the selected item. Use the [Value](#) property to select a value in a single column control. The [SelectableItem](#) property specifies whether the user can select an item. Use the [SelectItem](#) or [Select](#) property to select an item. The [RemoveSelection](#) method removes the selected items (including the descendents).

property Items.VisibleCount as Long

Retrieves the number of items being visible in the control's client area.

Type	Description
Long	Counts the visible items.

The VisibleCount property retrieves the number of items being displayed in the control's client area. Use the [VisibleItemCount](#) property to retrieve the number of visible items in the full list. 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 [ItemCount](#) property to count the items in the control. Use the [ChildCount](#) property to count the child items.

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 minus 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 [ItemCount](#) property to count the items in the control. Use the [ChildCount](#) property to count the child items.

UnboundHandler object

The control supports unbound mode. In unbound mode, the user is responsible for retrieving items. The unbound mode was 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.

Here's the IDL definition of the IUnboundHandler interface:

```
[
    uuid(84B7F083-9CA4-4C28-A569-CD38DF32C987),
    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, long ItemHandle );
}
```

Here's the IDL definition of the UnboundHandler interface (this interface is available starting from the version 12.0):

```
[
    uuid(84B7F083-9CA4-4C28-A569-CD38DF32C988),
]
dispinterface UnboundHandler
{
    interface IUnboundHandler;
}
```

The following **VB** sample displays 20,000 items in unbound mode:

```
Implements EXCOMBOBOXLibCtl.IUnboundHandler

Private Sub Form_Load()
    With ComboBox1
```

```

.BeginUpdate
.Columns.Add("Index").FormatColumn = "value format `0`"
Set .UnboundHandler = Me
.EndUpdate
End With
End Sub

Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
    IUnboundHandler_ItemsCount = 20000
End Property

Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object,
ByVal ItemHandle As Long)
    With Source.Items
        .CellCaption(ItemHandle, 0) = Index + 1
    End With
End Sub

```

The following **VB/NET** sample displays 20,000 items in unbound mode:

```

Public Class Form1
    Implements EXCOMBOBOXLib.IUnboundHandler

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        With AxComboBox1
            .BeginUpdate()
            .Columns.Add("Index").FormatColumn = "value format `0`"
            .UnboundHandler = Me
            .EndUpdate()
        End With
    End Sub

    Public ReadOnly Property ItemsCount(ByVal Source As Object) As Integer Implements
EXCOMBOBOXLib.IUnboundHandler.ItemsCount
        Get
            ItemsCount = 20000
        End Get
    End Property

```

```
End Get
End Property
```

```
Public Sub ReadItem(ByVal Index As Integer, ByVal Source As Object, ByVal ItemHandle
As Integer) Implements EXCOMBOBOXLib.IUnboundHandler.ReadItem
    With Source.Items
        .CellCaption(ItemHandle, 0) = Index + 1
    End With
End Sub
End Class
```

The following **C#** sample displays 20,000 items in unbound mode:

```
public partial class Form1 : Form, EXCOMBOBOXLib.IUnboundHandler
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        axComboBox1.BeginUpdate();
        (axComboBox1.Columns.Add("Index") as EXCOMBOBOXLib.IColumn).FormatColumnn
= "value format `0`";
        axComboBox1.UnboundHandler = this;
        axComboBox1.EndUpdate();
    }

    public int get_ItemsCount(object Source)
    {
        return 1000000;
    }

    public void ReadItem(int Index, object Source, int ItemHandle)
    {
        (Source as EXCOMBOBOXLib.IComboBox).Items.set_CellCaption(ItemHandle, 0, Index
```

```
+ 1);  
}  
}
```

The following **VFP 9** sample displays 20,000 items in unbound mode:

```
with thisform.ComboBox1  
    .Columns.Add("Index").FormatColumn = "value format `0`"  
    .UnboundHandler = newobject('UnboundHandler', 'class1.prg')  
endwith
```

where the class1.prg is:

```
define class UnboundHandler as session OLEPUBLIC  
  
implements IUnboundHandler in "ExComboBox.dll"  
function IUnboundHandler_get_ItemsCount(Source)  
    return 1000000  
endfunc  
function IUnboundHandler_ReadItem(Index, Source, ItemHandle)  
    With Source.Items  
        .CellCaption(ItemHandle, 0) = Index + 1  
    EndWith  
endfunc  
  
implements UnboundHandler in "ExComboBox.dll"  
function UnboundHandler_get_ItemsCount(Source)  
    return this.IUnboundHandler_get_ItemsCount(Source)  
endfunc  
function UnboundHandler_ReadItem(Index, Source, ItemHandle)  
    return this.IUnboundHandler_ReadItem(Index, Source, ItemHandle)  
endfunc  
  
enddefine
```

The following **VFP 7** and **VFP 8** sample displays 20,000 items in unbound mode:

```
with thisform.ComboBox1
```

```
.Columns.Add("Index").FormatColumn = "value format `0`"
```

```
.UnboundHandler = newobject('UnboundHandler', 'class1.prg')
```

```
endwith
```

where the class1.prg is:

```
define class UnboundHandler as custom
```

```
implements IUnboundHandler in "ExComboBox.dll"
```

```
function IUnboundHandler_get_ItemsCount(Source)
```

```
    return 20000
```

```
endfunc
```

```
function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
```

```
    With Source.Items
```

```
        .DefaultItem = ItemHandle
```

```
        .CellCaption(0, 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 mode.

The ItemsCount property specifies the number of items in unbound mode.

The following **VB** sample shows how ItemsCount property can be implemented:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
    IUnboundHandler_ItemsCount = 20000
End Property
```

The following **VB/NET** sample shows how ItemsCount property can be implemented:

```
Public ReadOnly Property ItemsCount(ByVal Source As Object) As Integer Implements
EXCOMBOBOXLib.IUnboundHandler.ItemsCount
    Get
        ItemsCount = 20000
    End Get
End Property
```

The following **C#** sample shows how ItemsCount property can be implemented:

```
public int get_ItemsCount(object Source)
{
    return 1000000;
}
```

The following **VFP** sample shows how ItemsCount property can be implemented:

```
function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
    With Source.Items
        .CellCaption(ItemHandle, 0) = Index + 1
    EndWith
endfunc
```

method UnboundHandler.ReadItem (Index as Long, Source as Object, ItemHandle as Long)

The source requires an item.

Type	Description
Index as Long	A Long expression that specifies the index of the item to be requested
Source as Object	The source object to be filled.
ItemHandle as Long	A Long expression that specifies the handle of the item in the source, that's associated with the requested index.

The ReadItem method is called every time the Source requires an item to be displayed.

The following **VB** sample shows how ReadItem could be implemented:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object,
ByVal ItemHandle As Long)
    With Source.Items
        .CellCaption(ItemHandle, 0) = Index + 1
    End With
End Sub
```

The following **VB/NET** sample shows how ReadItem could be implemented:

```
Public Sub ReadItem(ByVal Index As Integer, ByVal Source As Object, ByVal ItemHandle As
Integer) Implements EXCOMBOBOXLib.IUnboundHandler.ReadItem
    With Source.Items
        .CellCaption(ItemHandle, 0) = Index + 1
    End With
End Sub
```

The following **C#** sample shows how ReadItem could be implemented:

```
public void ReadItem(int Index, object Source, int ItemHandle)
{
    (Source as EXCOMBOBOXLib.IComboBox).Items.set_CellCaption(ItemHandle, 0, Index +
1);
}
```

The following **VFP** sample shows how ReadItem could be implemented:

```
function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
  With Source.Items
    .CellCaption(ItemHandle, 0) = Index + 1
  EndWith
endfunc
```

ExComboBox events

The ExComboBox component supports the following events:

Name	Description
AddColumn	Fired after a new column has been added.
AfterExpandItem	Fired after an item is expanded (collapsed).
AnchorClick	Occurs when an anchor element is clicked.
BeforeExpandItem	Fired before an item is about to be expanded (collapsed).
CellButtonClick	Fired after the user clicks on the cell of button type.
CellImageClick	Fired after the user clicks on the image's cell area.
CellStateChanged	Fired after cell's state has been changed.
CellStateChanging	Fired before cell's state is about to be changed.
Change	Occurs when user closes the drop down by selecting a value.
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.
DbClick	Occurs when the user dblclk the left mouse button over an object.
DropDown	Occurs when the drop-down portion of the control is shown.
DropUp	Occurs when the drop-down portion of the control is hidden.
EditChange	Fired when the user has taken an action that may have altered text in an edit control.
Event	Notifies the application once the control fires an event.
FilterChange	Occurs when filter was changed.
FormatColumn	Fired when a cell requires to format its caption.
InsertItem	Occurs after a new item has been inserted to Items collection.
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.
	Occurs when the user releases a key while an object has

KeyUp	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.
NotInList	Occurs when the user enters a value in the text box portion of a combo box that isn't in the combo box list.
OffsetChanged	Occurs when the scroll position has been changed.
OversizeChanged	Occurs when the right range of the scroll has been changed.
RClick	Fired when right mouse button is clicked
RemoveColumn	Fired before deleting a Column.
RemoveItem	Occurs before deleting an item.
ScrollBarClick	Occurs when the user clicks a button in the scrollbar.
SelectionChanged	Fired after a new item has been 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 has been added to Columns collection.

The AddColumn event is fired when a new column is inserted to the [Columns](#) collection. Use the [Add](#) method to add new columns to the control. Use the AddColumn event to associate extra data to a column. Use the [Data](#) property to assign an extra data to a column. The control fires the [RemoveColumn](#) event when a column is removed. Use the [ColumnAutoSize](#) property to specify whether the visible columns hits the control's client area.

The following VB sample sets the width for all columns:

```
Private Sub ComboBox1_AddColumn(ByVal Column As EXCOMBOBOXLibCtl.IColumn)
    Column.Width = 128
End Sub
```

The following C++ sample sets the width for all columns:

```
#include "Column.h"
void OnAddColumnCombobox1(LPDISPATCH Column)
{
    CColumn column( Column );column.m_bAutoRelease = FALSE;
    column.SetWidth( 128 );
}
```

The following VB.NET sample changes the column's width:

```
Private Sub AxComboBox1_AddColumn(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_AddColumnEvent) Handles
AxComboBox1.AddColumn
    e.column.Width = 128
End Sub
```

The following C# sample changes the column's width:

```
private void axComboBox1_AddColumn(object sender,
```

```
AxEXCOMBOBOXLib._IComboBoxEvents_AddColumnEvent e)
```

```
{  
    e.column.Width = 128;  
}
```

The following VFP sample changes the column's width:

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

```
LPARAMETERS column
```

```
with column
```

```
    .Width = 128
```

```
endwith
```

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

```
C# private void AddColumn(object sender,exontrol.EXCOMBOBOXLib.Column  
Column)  
{  
}
```

```
VB Private Sub AddColumn(ByVal sender As System.Object,ByVal Column As  
exontrol.EXCOMBOBOXLib.Column) Handles AddColumn  
End Sub
```

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

```
C# private void AddColumn(object sender,  
AxEXCOMBOBOXLib._IComboBoxEvents_AddColumnEvent e)  
{  
}
```

```
C++ void OnAddColumn(LPDISPATCH Column)  
{  
}
```

```
C++  
Builder void __fastcall AddColumn(TObject *Sender,Excomboboxlib_tlb::IColumn  
*Column)  
{
```

```
}
```

```
Delphi procedure AddColumn(ASender: TObject; Column : IColumn);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure AddColumn(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_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  
AxEXCOMBOBOXLib._IComboBoxEvents_AddColumnEvent) Handles AddColumn  
End Sub
```

```
VB6 Private Sub AddColumn(ByVal Column As EXCOMBOBOXLibCtl.IColumn)  
End Sub
```

```
VBA Private Sub AddColumn(ByVal Column As Object)  
End Sub
```

```
VFP LPARAMETERS Column
```

```
Xbas... PROCEDURE OnAddColumn(oComboBox,Column)  
RETURN
```

Syntax for AddColumn event, **ICOM** 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 IIColumn  
    Forward Send OnComAddColumn IIColumn  
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.ComboBox.1::IIColumn)  
end function
```

dBASE

```
function nativeObject_AddColumn(Column)  
return
```

event AfterExpandItem (Item as HITEM)

Fired after an item is expanded (collapsed).

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being expanded or collapsed.

The AfterExpandItem event notifies your application that an item is collapsed or expanded. Use the [ExpandItem](#) method to programmatically expand or collapse an item. The ExpandItem property also specifies whether an item is expanded or collapsed. The [ItemChild](#) property retrieves the first child item. Use the [BeforeExpandItem](#) event to cancel expanding or collapsing items.

The following VB sample prints the item's state when it is expanded or collapsed:

```
Private Sub ComboBox1_AfterExpandItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    Debug.Print "The " & Item & " item was " & If(ComboBox1.Items.ExpandItem(Item),
    "expanded", "collapsed")
End Sub
```

The following C++ sample prints the item's state when it is expanded or collapsed:

```
void OnAfterExpandItemCombobox1(long Item)
{
    if ( ::IsWindow( m_combobox.m_hWnd ) )
    {
        CItems items = m_combobox.GetItems();
        CString strFormat;
        strFormat.Format( "%s", items.GetExpandItem( Item ) ? "expanded" : "collapsed" );
        OutputDebugString( strFormat );
    }
}
```

The following C# sample prints the item's state when it is expanded or collapsed:

```
private void axComboBox1_AfterExpandItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_AfterExpandItemEvent e)
{
    System.Diagnostics.Debug.WriteLine(axComboBox1.Items.get_ExpandItem(e.item) ?
```

```
"expanded" : "collapsed");  
}
```

The following VB.NET sample prints the item's state when it is expanded or collapsed:

```
Private Sub AxComboBox1_AfterExpandItem(ByVal sender As Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_AfterExpandItemEvent) Handles  
AxComboBox1.AfterExpandItem  
    Debug.WriteLine(If(AxComboBox1.Items.ExpandItem(e.item), "expanded", "collapsed"))  
End Sub
```

The following VFP sample prints the item's state when it is expanded or collapsed:

```
*** ActiveX Control Event ***  
LPARAMETERS item  
  
with thisform.ComboBox1.Items  
    if ( .ExpandItem(item) )  
        wait window "expanded" nowait  
    else  
        wait window "collapsed" nowait  
    endif  
endwith
```

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

```
C# private void AfterExpandItem(object sender,int Item)  
{  
}
```

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

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

```
C# private void AfterExpandItem(object sender,  
AxEXCOMBOBOXLib._IComboBoxEvents_AfterExpandItemEvent e)
```

```
{  
}
```

```
C++  
void OnAfterExpandItem(long Item)  
{  
}
```

```
C++  
Builder  
void __fastcall AfterExpandItem(TObject *Sender,Excomboboxlib_tlb::HITEM Item)  
{  
}
```

```
Delphi  
procedure AfterExpandItem(ASender: TObject; Item : HITEM);  
begin  
end;
```

```
Delphi 8  
(.NET  
only)  
procedure AfterExpandItem(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_AfterExpandItemEvent);  
begin  
end;
```

```
Powe...  
begin event AfterExpandItem(long Item)  
end event AfterExpandItem
```

```
VB.NET  
Private Sub AfterExpandItem(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_AfterExpandItemEvent) Handles  
AfterExpandItem  
End Sub
```

```
VB6  
Private Sub AfterExpandItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)  
End Sub
```

```
VBA  
Private Sub AfterExpandItem(ByVal Item As Long)  
End Sub
```

```
VFP  
LPARAMETERS Item
```

```
Xbas...  
PROCEDURE OnAfterExpandItem(oComboBox,Item)
```

Syntax for AfterExpandItem event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="AfterExpandItem(Item)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function AfterExpandItem(Item)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComAfterExpandItem HITEM IItem  
Forward Send OnComAfterExpandItem IItem  
End_Procedure
```

```
Visual  
Objects METHOD OCX_AfterExpandItem(Item) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_AfterExpandItem(int _Item)  
{  
}
```

```
XBasic function AfterExpandItem as v (Item as OLE::Exontrol.ComboBox.1::HITEM)  
end function
```

```
dBASE function nativeObject_AfterExpandItem(Item)  
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 `<a1;youreextradata>anchor`, the **AnchorID** parameter of the **AnchorClick** event is 1, and the **Options** parameter is "youreextradata". Use the [AnchorFromPoint](#) property to retrieve the identifier of the anchor element from the cursor.

Syntax for **AnchorClick** event, **/.NET** version, on:

```
C# private void AnchorClick(object sender,string AnchorID,string Options)
{
}
```

```
VB Private Sub AnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles AnchorClick
End Sub
```

Syntax for **AnchorClick** event, **/.COM** version, on:

```
C# private void AnchorClick(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_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:
AxEXCOMBOBOXLib._IComboBoxEvents_AnchorClickEvent);
begin
end;
```

Power...

```
begin event AnchorClick(string AnchorID,string Options)
end event AnchorClick
```

VB.NET

```
Private Sub AnchorClick(ByVal sender As System.Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,AnchorID,Options)
RETURN
```

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

```
Java... <SCRIPT EVENT="AnchorClick(AnchorID,Options)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function AnchorClick(AnchorID,Options)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComAnchorClick String IIAnchorID String IIOptions  
Forward Send OnComAnchorClick IIAnchorID IIOptions  
End_Procedure
```

```
Visual  
Objects METHOD OCX_AnchorClick(AnchorID,Options) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_AnchorClick(str _AnchorID,str _Options)  
{  
}
```

```
XBasic function AnchorClick as v (AnchorID as C,Options as C)  
end function
```

```
dBASE function nativeObject_AnchorClick(AnchorID,Options)  
return
```

event BeforeExpandItem (Item as HITEM, Cancel as Variant)

Fired before an item is about to be expanded (collapsed).

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being expanded or collapsed.
Cancel as Variant	A boolean expression that indicates whether the user cancels the expanding/collapsing operation.

The BeforeExpandItem event notifies your application that an item is about to be collapsed or expanded. Use the BeforeExpandItem event to cancel expanding or collapsing items. Use the BeforeExpandItem event to load new items when filling a virtual tree. The [AfterExpandItem](#) event is fired after an item is expanded or collapsed. Use the [ExpandItem](#) method to programmatically expand or collapse an item. Use the [ExpandOnSearch](#) property to expand items while user types characters to search for items using incremental search feature.

The following VB sample cancels expanding or collapsing items:

```
Private Sub ComboBox1_BeforeExpandItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM,
Cancel As Variant)
    Cancel = True
End Sub
```

The following C++ sample cancels expanding or collapsing items:

```
void OnBeforeExpandItemCombobox1(long Item, VARIANT FAR* Cancel)
{
    V_VT( Cancel ) = VT_BOOL;
    V_BOOL( Cancel ) = VARIANT_TRUE;
}
```

The following C# sample cancels expanding or collapsing items:

```
private void axComboBox1_BeforeExpandItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent e)
{
    e.cancel = true;
}
```

The following VB.NET sample cancels expanding or collapsing items:

```
Private Sub AxComboBox1_BeforeExpandItem(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent) Handles
AxComboBox1.BeforeExpandItem
    e.cancel = True
End Sub
```

The following VFP sample cancels expanding or collapsing items:

```
*** ActiveX Control Event ***
LPARAMETERS item, cancel

cancel = .t.
```

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

```
C# private void BeforeExpandItem(object sender,int Item,ref object Cancel)
{
}
```

```
VB Private Sub BeforeExpandItem(ByVal sender As System.Object,ByVal Item As
Integer,ByRef Cancel As Object) Handles BeforeExpandItem
End Sub
```

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

```
C# private void BeforeExpandItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent e)
{
}
```

```
C++ void OnBeforeExpandItem(long Item,VARIANT FAR* Cancel)
{
}
```

```
C++
Builder void __fastcall BeforeExpandItem(TObject *Sender,Excomboboxlib_tlb::HITEM
Item,Variant * Cancel)
{
```

```
}
```

```
Delphi procedure BeforeExpandItem(ASender: TObject; Item : HITEM;var Cancel :  
OleVariant);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure BeforeExpandItem(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent);  
begin  
end;
```

```
Powe... begin event BeforeExpandItem(long Item,any Cancel)  
end event BeforeExpandItem
```

```
VB.NET Private Sub BeforeExpandItem(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_BeforeExpandItemEvent) Handles  
BeforeExpandItem  
End Sub
```

```
VB6 Private Sub BeforeExpandItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM,Cancel  
As Variant)  
End Sub
```

```
VBA Private Sub BeforeExpandItem(ByVal Item As Long,Cancel As Variant)  
End Sub
```

```
VFP LPARAMETERS Item,Cancel
```

```
Xbas... PROCEDURE OnBeforeExpandItem(oComboBox,Item,Cancel)  
RETURN
```

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

```
Java... <SCRIPT EVENT="BeforeExpandItem(Item,Cancel)" LANGUAGE="JScript">  
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">  
Function BeforeExpandItem(Item,Cancel)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComBeforeExpandItem HITEM IItem Variant IICancel  
Forward Send OnComBeforeExpandItem IItem IICancel  
End_Procedure
```

Visual
Objects

```
METHOD OCX_BeforeExpandItem(Item,Cancel) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_BeforeExpandItem(int _Item,COMVariant /*variant*/ _Cancel)  
{  
}
```

XBasic

```
function BeforeExpandItem as v (Item as OLE::Exontrol.ComboBox.1::HITEM,Cancel  
as A)  
end function
```

dBASE

```
function nativeObject_BeforeExpandItem(Item,Cancel)  
return
```

event CellButtonClick (Cell as HCELL)

Fired after the user clicks on the cell of button type.

Type	Description
Cell as HCELL	A long expression that indicates the handle of the cell being clicked.

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 [CellCaption](#) property to determine the caption of the cell being clicked.

The following VB sample sets the cells of the first column to be of button type, and displays a message when one of them has been clicked.

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    ComboBox1.Items.CellHasButton(Item, 0) = True
End Sub

Private Sub ComboBox1_CellButtonClick(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    MsgBox "The user clicks the cell's button. " & ComboBox1.Items.CellCaption(, Cell)
End Sub
```

The following C++ sample displays a message when the user clicks a button in the cell:

```
void OnCellButtonClickCombobox1(long Cell)
{
    COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
    CString strCaption = V2S( &m_combobox.GetItems().GetCellCaption( vtMissing,
COleVariant( (long)Cell ) ) );
    MessageBox( "The cell of button type has been clicked." + 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 a message when the user clicks a button in the cell:

```

Private Sub AxComboBox1_CellButtonClick(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_CellButtonClickEvent) Handles
AxComboBox1.CellButtonClick
    MsgBox("The cell of button type has been clicked." & AxComboBox1.Items.CellCaption(
e.cell))
End Sub

```

The following C# sample displays a message when the user clicks a button in the cell:

```

private void axComboBox1_CellButtonClick(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_CellButtonClickEvent e)
{
    MessageBox.Show("The cell of button type has been clicked. " +
axComboBox1.Items.get_CellCaption(null, e.cell).ToString() );
}

```

The following VFP sample displays a message when the user clicks a button in the cell:

```

*** ActiveX Control Event ***
LPARAMETERS item, colindex

wait window "The cell of button type has been clicked."

```

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

```

C# private void CellButtonClick(object sender,exontrol.EXCOMBOBOXLib.HCELL Cell)
{

```

```
}
```

VB

```
Private Sub CellButtonClick(ByVal sender As System.Object,ByVal Cell As  
exontrol.EXCOMBOBOXLib.HCELL) Handles CellButtonClick  
End Sub
```

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

C#

```
private void CellButtonClick(object sender,  
AxEXCOMBOBOXLib._IComboBoxEvents_CellButtonClickEvent e)  
{  
}
```

C++

```
void OnCellButtonClick(long Cell)  
{  
}
```

C++**Builder**

```
void __fastcall CellButtonClick(TObject *Sender,Excomboboxlib_tlb::HCELL Cell)  
{  
}
```

Delphi

```
procedure CellButtonClick(ASender: TObject; Cell : HCELL);  
begin  
end;
```

Delphi 8**(.NET
only)**

```
procedure CellButtonClick(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_CellButtonClickEvent);  
begin  
end;
```

Powe...

```
begin event CellButtonClick(long Cell)  
end event CellButtonClick
```

VB.NET

```
Private Sub CellButtonClick(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_CellButtonClickEvent) Handles  
CellButtonClick  
End Sub
```

VB6

```
Private Sub CellButtonClick(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
End Sub
```

VBA

```
Private Sub CellButtonClick(ByVal Cell As Long)
End Sub
```

VFP

```
LPARAMETERS Cell
```

Xbas...

```
PROCEDURE OnCellButtonClick(oComboBox,Cell)
RETURN
```

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

Java...

```
<SCRIPT EVENT="CellButtonClick(Cell)" LANGUAGE="JScript">
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">
Function CellButtonClick(Cell)
End Function
</SCRIPT>
```

Visual
Data...

```
Procedure OnComCellButtonClick HCELL IICell
Forward Send OnComCellButtonClick IICell
End_Procedure
```

Visual
Objects

```
METHOD OCX_CellButtonClick(Cell) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellButtonClick(int _Cell)
{
}
```

XBasic

```
function CellButtonClick as v (Cell as OLE::Exontrol.ComboBox.1::HCELL)
end function
```

```
function nativeObject_CellButtonClick(Cell)
return
```

event CellImageClick (Cell as HCELL)

Fired after the user clicks on the image's cell area.

Type	Description
Cell as HCELL	A long expression that indicates the handle of the cell where the user clicks the cell's image.

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. The following sample changes the cell's image when the user has clicked on the cell's image (to run the following sample you have to add two images to the tree's images collection.). 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.

The following VB sample assigns an icon to each cell that's added, and changes the cell's icon when the user clicks the icon:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    ComboBox1.Items.CellImage(Item, 0) = 1
End Sub

Private Sub ComboBox1_CellImageClick(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    ComboBox1.Items.CellImage(, Cell) = ComboBox1.Items.CellImage(, Cell) Mod 2 + 1
End Sub
```

The following VB sample displays the index of icon being clicked:

```
Private Sub ComboBox1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim i As HITEM, h As HitTestInfoEnum, c As Long
    With ComboBox1
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
    End With
    If (i <> 0) Or (c <> 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 If
End Sub
```

The following C++ sample changes the cell's icon being clicked:

```
#include "items.h"
void OnCellImageClickCombobox1(long Cell)
{
    CItems items = m_combobox.GetItems();
    COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
    COleVariant vtColumn( Cell );
    items.SetCellImage( vtMissing , vtColumn , items.GetCellImage( vtMissing, vtColumn ) %
2 + 1 );
}
```

The following C# sample changes the cell's icon being clicked:

```
private void axComboBox1_CellImageClick(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_CellImageClickEvent e)
{
    axComboBox1.Items.set_CellImage(null, e.cell, axComboBox1.Items.get_CellImage(null,
e.cell) % 2 + 1);
}
```

The following VB/NET sample changes the cell's icon being clicked:

```
Private Sub AxComboBox1_CellImageClick(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_CellImageClickEvent) Handles
AxComboBox1.CellImageClick
    With AxComboBox1.Items
        .CellImage(Nothing, e.cell) = .CellImage(Nothing, e.cell) Mod 2 + 1
    End With
End Sub
```

The following VFP sample changes the cell's icon being clicked:

```
*** ActiveX Control Event ***
LPARAMETERS cell
```

```
with thisform.ComboBox1.Items
    .DefaultItem = .CellItem(cell)
    .CellImage( 0,0 ) = .CellImage( 0,0 ) + 1
endwith
```

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

```
C# private void CellImageClick(object sender,exontrol.EXCOMBOBOXLib.HCELL Cell)
{
}
```

```
VB Private Sub CellImageClick(ByVal sender As System.Object,ByVal Cell As
exontrol.EXCOMBOBOXLib.HCELL) Handles CellImageClick
End Sub
```

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

```
C# private void CellImageClick(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_CellImageClickEvent e)
{
}
```

```
C++ void OnCellImageClick(long Cell)
{
}
```

```
C++ Builder void __fastcall CellImageClick(TObject *Sender,Excomboboxlib_tlb::HCELL Cell)
{
}
```

```
Delphi procedure CellImageClick(ASender: TObject; Cell : HCELL);
begin
end;
```

```
Delphi 8 (.NET only) procedure CellImageClick(sender: System.Object; e:
AxEXCOMBOBOXLib._IComboBoxEvents_CellImageClickEvent);
begin
```

```
end;
```

```
Powe... begin event CellImageClick(long Cell)  
end event CellImageClick
```

```
VB.NET Private Sub CellImageClick(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_CellImageClickEvent) Handles  
CellImageClick  
End Sub
```

```
VB6 Private Sub CellImageClick(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)  
End Sub
```

```
VBA Private Sub CellImageClick(ByVal Cell As Long)  
End Sub
```

```
VFP LPARAMETERS Cell
```

```
Xbas... PROCEDURE OnCellImageClick(oComboBox,Cell)  
RETURN
```

Syntax for CellImageClick event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="CellImageClick(Cell)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function CellImageClick(Cell)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComCellImageClick HCELL IICell  
Forward Send OnComCellImageClick IICell  
End_Procedure
```

```
Visual  
Objects METHOD OCX_CellImageClick(Cell) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_CellImageClick(int _Cell)
```

```
{  
}
```

XBasic

```
function CellImageClick as v (Cell as OLE::Exontrol.ComboBox.1::HCELL)  
end function
```

dBASE

```
function nativeObject_CellImageClick(Cell)  
return
```

event CellStateChanged (Cell as HCELL)

Fired after cell's state has been changed.

Type	Description
Cell as HCELL	A long expression that indicates the handle of the cell whose state is changed.

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.

The following VB sample displays a message when the user clicks a check box or a radio button:

```
Private Sub ComboBox1_CellStateChanged(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)
    With ComboBox1.Items
        Debug.Print "The cell """" & .CellCaption(, Cell) & """" has changed its state. The new
state is " & If(.CellState(, Cell) = 0, "Unchecked", "Checked")
    End With
End Sub
```

The following VC sample displays the caption of the cell whose checkbox's state is changed:

```
#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 OnCellStateChangedCombobox1(long Cell)
{
    if ( ::IsWindow( m_combobox.m_hWnd ) )
    {
        CItems items = m_combobox.GetItems();
        COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
        COleVariant vtCell( Cell );
        CString strCellCaption = V2S( &items.GetCellCaption( vtMissing, vtCell ) );
        CString strOutput;
        strOutput.Format( "%s's checkbox state is %i\r\n", strCellCaption, items.GetCellState(
vtMissing, vtCell ) );
        OutputDebugString( strOutput );
    }
}
}

```

The following VB.NET sample displays a message when the user clicks a check box or a radio button:

```

Private Sub AxComboBox1_CellStateChanged(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent) Handles
AxComboBox1.CellStateChanged
    With AxComboBox1.Items
        Debug.Print("The cell """" & .CellCaption(, e.cell) & """" has changed its state. The new
state is " & If(.CellState(, e.cell) = 0, "Unchecked", "Checked"))
    End With
End Sub

```

The following C# sample outputs a message when the user clicks a check box or a radio button:

```

private void axComboBox1_CellStateChanged(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent e)
{
    string strOutput = axComboBox1.Items.get_CellCaption(null, e.cell).ToString();
}

```

```
strOutput += " state = " + axComboBox1.Items.get_CellState(null, e.cell).ToString();
System.Diagnostics.Debug.WriteLine(strOutput);
}
```

The following VFP sample prints a message when the user clicks a check box or a radio button:

```
*** ActiveX Control Event ***
LPARAMETERS cell

local sOutput
sOutput = ""
with thisform.ComboBox1.Items
    .DefaultItem = .CellItem(cell)
    sOutput = .CellCaption( 0, 0 )
    sOutput = sOutput + ", state = " + str(.CellState( 0, 0 ))
    wait window nowait sOutput
endwith
```

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

```
C# private void CellStateChanged(object sender,exontrol.EXCOMBOBOXLib.HCELL
Cell)
{
}
```

```
VB Private Sub CellStateChanged(ByVal sender As System.Object,ByVal Cell As
exontrol.EXCOMBOBOXLib.HCELL) Handles CellStateChanged
End Sub
```

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

```
C# private void CellStateChanged(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent e)
{
}
```

```
C++ void OnCellStateChanged(long Cell)
```

```
{  
}
```

C++
Builder

```
void __fastcall CellStateChanged(TObject *Sender,Excomboboxlib_tlb::HCELL Cell)  
{  
}
```

Delphi

```
procedure CellStateChanged(ASender: TObject; Cell : HCELL);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure CellStateChanged(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent);  
begin  
end;
```

Power...

```
begin event CellStateChanged(long Cell)  
end event CellStateChanged
```

VB.NET

```
Private Sub CellStateChanged(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangedEvent) Handles  
CellStateChanged  
End Sub
```

VB6

```
Private Sub CellStateChanged(ByVal Cell As EXCOMBOBOXLibCtl.HCELL)  
End Sub
```

VBA

```
Private Sub CellStateChanged(ByVal Cell As Long)  
End Sub
```

VFP

```
LPARAMETERS Cell
```

Xbas...

```
PROCEDURE OnCellStateChanged(oComboBox,Cell)  
RETURN
```

Syntax for CellStateChanged event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="CellStateChanged(Cell)" LANGUAGE="JScript">  
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">  
Function CellStateChanged(Cell)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComCellStateChanged HCELL IICell  
    Forward Send OnComCellStateChanged IICell  
End_Procedure
```

Visual
Objects

```
METHOD OCX_CellStateChanged(Cell) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_CellStateChanged(int _Cell)  
{  
}
```

XBasic

```
function CellStateChanged as v (Cell as OLE::Exontrol.ComboBox.1::HCELL)  
end function
```

dBASE

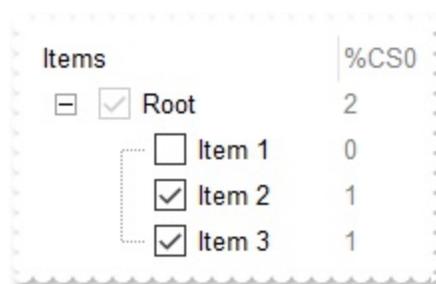
```
function nativeObject_CellStateChanged(Cell)  
return
```

event CellStateChanging (Cell as HCELL, NewState as Long)

Fired before cell's state is about to be changed.

Type	Description
Cell as HCELL	A long expression that indicates the handle of the cell to change its state is about to be changed.
NewState as Long	A long expression that specifies the new state of the cell (0- unchecked, 1 - checked, 2 - partial 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(,Cell)`. 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 [PartialCheck](#) property to enable partial check feature (check boxes with three states: partial, checked and unchecked). 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 ComboBox1_CellStateChanging(ByVal Cell As EXCOMBOBOXLibCtl.HCELL,
NewState As Long)
    With ComboBox1
        If (.ReadOnly) Then
            With .Items
                NewState = .CellState(, Cell)
            End With
        End If
    End With
End Sub

```

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

```

C# private void CellStateChanging(object sender,int Cell,ref int NewState)
{
}

```

```

VB Private Sub CellStateChanging(ByVal sender As System.Object,ByVal Cell As
Integer,ByRef NewState As Integer) Handles CellStateChanging
End Sub

```

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

```

C# private void CellStateChanging(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangingEvent e)
{
}

```

```

C++ void OnCellStateChanging(long Cell,long FAR* NewState)
{
}

```

```

C++ Builder void __fastcall CellStateChanging(TObject *Sender,Extreelib_tlb::HCELL Cell,long *
NewState)
{
}

```

```

Delphi procedure CellStateChanging(ASender: TObject; Cell: HCELL;var NewState :
Integer);

```

```
begin  
end;
```

```
Delphi 8  
(.NET  
only)  
procedure CellStateChanging(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangingEvent);  
begin  
end;
```

```
Power...  
begin event CellStateChanging(long Cell,long NewState)  
  
end event CellStateChanging
```

```
VB.NET  
Private Sub CellStateChanging(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_CellStateChangingEvent) Handles  
CellStateChanging  
End Sub
```

```
VB6  
Private Sub CellStateChanging(ByVal Cell As EXCOMBOBOXLibCtl.HCELL,NewState  
As Long)  
End Sub
```

```
VBA  
Private Sub CellStateChanging(ByVal Cell As Long,NewState As Long)  
End Sub
```

```
VFP  
LPARAMETERS Cell,NewState
```

```
Xbas...  
PROCEDURE OnCellStateChanging(oComboBox,Cell,NewState)  
  
RETURN
```

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

```
Java...  
<SCRIPT EVENT="CellStateChanging(Cell,NewState)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS...  
<SCRIPT LANGUAGE="VBScript">  
Function CellStateChanging(Cell,NewState)
```

```
End Function  
</SCRIPT>
```

Visual Data...

```
Procedure OnComCellStateChanging HCELL IICell Integer IINewState  
    Forward Send OnComCellStateChanging IICell IINewState  
End_Procedure
```

Visual Objects

```
METHOD OCX_CellStateChanging(Cell,NewState) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_CellStateChanging(int _Cell,COMVariant /*long*/ _NewState)  
{  
}
```

XBasic

```
function CellStateChanging as v (Cell as  
OLE::Exontrol.ComboBox.1::HCELL,NewState as N)  
end function
```

dBASE

```
function nativeObject_CellStateChanging(Cell,NewState)  
return
```

event Change ()

Occurs when user closes the drop down by selecting a value.

Type

Description

The Change event is fired if the drop-down portion of the control is visible, and the user clicks an item or presses the Enter key. The Change event is not fired if the control's [Style](#) property is Simple. Use the [SelectionChange](#) event to notify your application that a new item is selected. Use the [EditText](#) property to get the caption of the edit inside the control's label. The [Select](#) property gets the value of selected item on a specified column, if the Style's property is DropDown or DropDownList. The [SelectableItem](#) property specifies whether the user can select an item.

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

```
C# private void Change(object sender)
{
}
```

```
VB Private Sub Change(ByVal sender As System.Object) Handles Change
End Sub
```

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

```
C# private void Change(object sender, EventArgs e)
{
}
```

```
C++ void OnChange()
{
}
```

```
C++ Builder void __fastcall Change(TObject *Sender)
{
}
```

```
Delphi procedure Change(ASender: TObject; );
begin
end;
```

Delphi 8
(.NET
only)

```
procedure Change(sender: System.Object; e: System.EventArgs);  
begin  
end;
```

Power...
begin event Change()
end event Change

VB.NET
Private Sub Change(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Change
End Sub

VB6
Private Sub Change()
End Sub

VBA
Private Sub Change()
End Sub

VFP
LPARAMETERS nop

Xbas...
PROCEDURE OnChange(oComboBox)
RETURN

Syntax for Change event, **ICOM** version (others), on:

Java...
<SCRIPT EVENT="Change()" LANGUAGE="JScript">
</SCRIPT>

VBSc...
<SCRIPT LANGUAGE="VBScript">
Function Change()
End Function
</SCRIPT>

Visual
Data...
Procedure OnComChange
Forward Send OnComChange

End_Procedure

Visual
Objects

```
METHOD OCX_Change() CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Change()  
{  
}
```

XBasic

```
function Change as v ()  
end function
```

dBASE

```
function nativeObject_Change()  
return
```

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, if the [FireClickOnSelect](#) property is False. If the FireClickOnSelect property is True, the Click event is fired when the user selects a new item in the control (using the keys or the mouse). 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. Use the [Change](#) event to notify your application that user clicks an item or presses the ENTER key. Use the [SelectionChanged](#) event to notify your application that a new item is selected.

The following VB sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
Private Sub ComboBox1_Click()  
    Debug.Print ComboBox1.Value  
End Sub
```

The following C++ sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```
void OnClickCombobox1()  
{  
    OutputDebugString( V2S( &m_combobox.GetValue() ) );  
}
```

where the V2S function converts a VARIANT value 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 the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```

Private Sub AxComboBox1_ClickEvent(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxComboBox1.ClickEvent
    Debug.WriteLine(AxComboBox1.Value)
End Sub

```

The following C# sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```

private void axComboBox1_ClickEvent(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axComboBox1.Value.ToString());
}

```

The following VFP sample displays the selected value as soon as the user selects a new item (the FireClickOnSelect property is True):

```

*** ActiveX Control Event ***

wait window nowait thisform.ComboBox1.Object.Value

```

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(oComboBox)
RETURN
```

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

```
Java... <SCRIPT EVENT="Click()" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <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 that specifies the column being clicked.

The ColumnClick event is fired when the user clicks the column's header. By default, the control sorts the column if the user clicks the column's header. Use the [SortChildren](#) property to sort the list by a column. Use the [ItemFromPoint](#) property to access the item from point. Use the [SortOnClick](#) property to disable sorting columns by clicking the control's header bar. Use the [HeaderVisible](#) property to specify whether the control's header bar is visible or hidden. Use the [Add](#) method to add new columns to the control.

The following VB sample prints the caption of the column being clicked:

```
Private Sub ComboBox1_ColumnClick(ByVal Column As EXCOMBOBOXLibCtl.IColumn)
    Debug.Print Column.Caption
End Sub
```

The following C++ sample prints the caption of the column being clicked:

```
#include "Column.h"
void OnColumnClickCombobox1(LPDISPATCH Column)
{
    CColumn column( Column );
    column.m_bAutoRelease = FALSE;
    OutputDebugString( column.GetCaption() );
}
```

The following VB.NET sample prints the caption of the column being clicked:

```
Private Sub AxComboBox1_ColumnClick(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_ColumnClickEvent) Handles
AxComboBox1.ColumnClick
    Debug.WriteLine(e.column.Caption)
End Sub
```

The following C# sample prints the caption of the column being clicked:

```
private void axComboBox1_ColumnClick(object sender,
```

```
AxEXCOMBOBOXLib._IComboBoxEvents_ColumnClickEvent e)
```

```
{  
    System.Diagnostics.Debug.WriteLine(e.column.Caption );  
}
```

The following VFP sample prints the caption of the column being clicked:

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

```
LPARAMETERS column
```

```
with column
```

```
    wait window nowait .Caption
```

```
endwith
```

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

```
C# private void ColumnClick(object sender,exontrol.EXCOMBOBOXLib.Column  
    Column)  
    {  
    }
```

```
VB Private Sub ColumnClick(ByVal sender As System.Object,ByVal Column As  
    exontrol.EXCOMBOBOXLib.Column) Handles ColumnClick  
    End Sub
```

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

```
C# private void ColumnClick(object sender,  
    AxEXCOMBOBOXLib._IComboBoxEvents_ColumnClickEvent e)  
    {  
    }
```

```
C++ void OnColumnClick(LPDISPATCH Column)  
    {  
    }
```

```
C++ Builder void __fastcall ColumnClick(TObject *Sender,Excomboboxlib_tlb::IColumn  
    *Column)  
    {
```

```
}
```

```
Delphi procedure ColumnClick(ASender: TObject; Column : IColumn);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure ColumnClick(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_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  
AxEXCOMBOBOXLib._IComboBoxEvents_ColumnClickEvent) Handles ColumnClick  
End Sub
```

```
VB6 Private Sub ColumnClick(ByVal Column As EXCOMBOBOXLibCtl.IColumn)  
End Sub
```

```
VBA Private Sub ColumnClick(ByVal Column As Object)  
End Sub
```

```
VFP LPARAMETERS Column
```

```
Xbas... PROCEDURE OnColumnClick(oComboBox,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 IIColumn  
    Forward Send OnComColumnClick IIColumn  
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.ComboBox.1::IIColumn)  
end function
```

dBASE

```
function nativeObject_ColumnClick(Column)  
return
```

event DbIcIck (Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user dblclk the left mouse button over an object.

Type	Description
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

The DbIcIck event is fired when the user dbl clicks on the control. Use the DbIcIck event to notify your application that a cell has been double-clicked. Use the [ItemFromPoint](#) property to determine the cell or the item over the cursor. Use the [CloseOnDbIcIck](#) property to specify whether the drop down portion of the control is hidden when the user double clicks an item. Use the [Style](#) property to specify whether the control shows a drop down portion.

The following VB sample prints the caption of the cell that's dbl clicked:

```
Private Sub ComboBox1_DbIcIck(Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        Dim i As HITEM, c As Long, hit As EXCOMBOBOXLibCtl.HitTestInfoEnum
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If (i >= 0) Then
            Debug.Print .Items.CellCaption(i, c) & " HT = " & hit
        End If
    End With
End Sub
```

The following C++ sample prints the caption of the cell that's dbl clicked:

```
void OnDbIcIckCombobox1(short Shift, long X, long Y)
{
    long c = 0, h = 0, i = m_combobox.GetItemFromPoint( X, Y, &c, &h );
    if ( i != 0 )
    {
```

```

CString strOutput;
strOutput.Format( "'%s' HT= %i\r\n", V2S(
&m_combobox.GetItems().GetCellCaption(COLEVariant(i), COleVariant(c) ), h );
OutputDebugString( strOutput );
}
}

```

where the V2S function converts a VARIANT 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 that's dbl clicked:

```

Private Sub AxComboBox1_DblClick(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib.IComboBoxEvents_DblClickEvent) Handles AxComboBox1.DblClick
    With AxComboBox1
        Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (i >= 0) Then
            Debug.WriteLine(.Items.CellCaption(i, c) & " HT = " & hit)
        End If
    End With
End Sub

```

The following C# sample prints the caption of the cell that's dbl clicked:

```

private void axComboBox1_DblClick(object sender,

```

```
AxEXCOMBOBOXLib._IComboBoxEvents_DblClickEvent e)
```

```
{  
    EXCOMBOBOXLib.HitTestInfoEnum h;  
    int c = 0, i = axComboBox1.get_ItemFromPoint(e.x, e.y, out c, out h);  
    if (i != 0)  
  
System.Diagnostics.Debug.WriteLine(axComboBox1.Items.get_CellCaption(i,c).ToString() +  
" HT = " + h.ToString() );  
}
```

The following VFP sample prints the caption of the cell that's dbl clicked:

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

```
LPARAMETERS shift, x, y
```

```
With thisform.ComboBox1
```

```
    local c, hit
```

```
    c = 0
```

```
    hit = 0
```

```
    .Items.DefaultItem = .ItemFromPoint(X , Y, @c, @hit)
```

```
    If (.Items.DefaultItem >= 0) Then
```

```
        wait window nowait .Items.CellCaption(0, c) + " HT = " + str(hit)
```

```
    EndIf
```

```
EndWith
```

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,  
AxEXCOMBOBOXLib._IComboBoxEvents_DblClickEvent e)
```

```
{  
}
```

```
C++ void OnDbClick(short Shift,long X,long Y)
```

```
{  
}
```

```
C++ Builder void __fastcall DbClick(TObject *Sender,short Shift,int X,int Y)
```

```
{  
}
```

```
Delphi procedure DbClick(ASender: TObject; Shift : Smallint;X : Integer;Y : Integer);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure DbClick(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_DblClickEvent);  
begin  
end;
```

```
Powe... begin event DbClick(integer Shift,long X,long Y)  
end event DbClick
```

```
VB.NET Private Sub DbClick(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_DblClickEvent) Handles DbClick  
End Sub
```

```
VB6 Private Sub DbClick(Shift As Integer,X As Single,Y As Single)  
End Sub
```

```
VBA Private Sub DbClick(ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)  
End Sub
```

```
VFP LPARAMETERS Shift,X,Y
```

```
Xbas... PROCEDURE OnDbClick(oComboBox,Shift,X,Y)  
RETURN
```

Syntax for DbIcClick event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="DbIcClick(Shift,X,Y)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function DbIcClick(Shift,X,Y)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComDbIcClick Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS  
IYY  
    Forward Send OnComDbIcClick IIShift IIX IYY  
End_Procedure
```

```
Visual  
Objects METHOD OCX_DbIcClick(Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_DbIcClick(int _Shift,int _X,int _Y)  
{  
}  
}
```

```
XBasic function DbIcClick as v (Shift as N,X as  
OLE::Exontrol.ComboBox.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.ComboBox.1::OLE_YPOS_PIXELS)  
end function
```

```
dBASE function nativeObject_DbIcClick(Shift,X,Y)  
return
```

event DropDown (Cancel as Variant)

Occurs when the drop-down portion of the control is shown.

Type	Description
Cancel as Variant	A Boolean expression that specifies whether to cancel showing the drop down portion of the control.

Use the DropDown and [DropUp](#) events to notify your application that the drop-down portion of the control is shown or hidden. Use the DropDown event to cancel showing the drop down portion of the control based on your condition. The DropDown and DropUp events are fired only if the control's [Style](#) property is DropDown or DropDownList. Use the [hWndDropDown](#) property to get the handle of the drop down window. Use the [DropDown](#) method to show programmatically the drop down portion of the control. The [AutoDropDown](#) property retrieves or sets a value that indicates whether the control's list automatically drops down once the user starts type into it.

The following VB sample disable showing the drop down portion of the control if there is no items:

```
Private Sub ComboBox1_DropDown(Cancel As Variant)
    Cancel = ComboBox1.Items.ItemCount = 0
End Sub
```

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

```
C# private void DropDown(object sender,ref object Cancel)
{
}
```

```
VB Private Sub DropDown(ByVal sender As System.Object,ByRef Cancel As Object)
Handles DropDown
End Sub
```

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

```
C# private void DropDown(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_DropDownEvent e)
{
}
```

C++

```
void OnDropDown(VARIANT FAR* Cancel)
{
}
```

**C++
Builder**

```
void __fastcall DropDown(TObject *Sender,Variant * Cancel)
{
}
```

Delphi

```
procedure DropDown(ASender: TObject; var Cancel : OleVariant);
begin
end;
```

**Delphi 8
(.NET
only)**

```
procedure DropDown(sender: System.Object; e:
AxEXCOMBOBOXLib._IComboBoxEvents_DropDownEvent);
begin
end;
```

Powe...

```
begin event DropDown(any Cancel)
end event DropDown
```

VB.NET

```
Private Sub DropDown(ByVal sender As System.Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_DropDownEvent) Handles DropDown
End Sub
```

VB6

```
Private Sub DropDown(Cancel As Variant)
End Sub
```

VBA

```
Private Sub DropDown(Cancel As Variant)
End Sub
```

VFP

```
LPARAMETERS Cancel
```

Xbas...

```
PROCEDURE OnDropDown(oComboBox,Cancel)
RETURN
```

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

```
Java... <SCRIPT EVENT="DropDown(Cancel)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function DropDown(Cancel)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComDropDown Variant IICancel  
Forward Send OnComDropDown IICancel  
End_Procedure
```

```
Visual  
Objects METHOD OCX_DropDown(Cancel) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_DropDown(COMVariant /*variant*/ _Cancel)  
{  
}
```

```
XBasic function DropDown as v (Cancel as A)  
end function
```

```
dBASE function nativeObject_DropDown(Cancel)  
return
```

event DropUp ()

Occurs when the drop-down portion of the control is hidden.

Type

Description

Use the [DropDown](#) and DropUp events to notify your application when the drop-down portion of the control is shown or hidden. The DropDown and DropUp events are fired only if the control's [Style](#) property is DropDown or DropDownList. Use the [hWndDropDown](#) property to get the handle of the drop down window. The [AutoDropDown](#) property retrieves or sets a value that indicates whether the control's list automatically drops down once the user starts type into it.

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

```
C# private void DropUp(object sender)
{
}
```

```
VB Private Sub DropUp(ByVal sender As System.Object) Handles DropUp
End Sub
```

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

```
C# private void DropUp(object sender, EventArgs e)
{
}
```

```
C++ void OnDropUp()
{
}
```

```
C++ Builder void __fastcall DropUp(TObject *Sender)
{
}
```

```
Delphi procedure DropUp(ASender: TObject; );
begin
end;
```

Delphi 8
(.NET
only)

```
procedure DropUp(sender: System.Object; e: System.EventArgs);  
begin  
end;
```

Power...
begin event DropUp()
end event DropUp

VB.NET
Private Sub DropUp(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles DropUp
End Sub

VB6
Private Sub DropUp()
End Sub

VBA
Private Sub DropUp()
End Sub

VFP
LPARAMETERS nop

Xbas...
PROCEDURE OnDropUp(oComboBox)
RETURN

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

Java...
<SCRIPT EVENT="DropUp()" LANGUAGE="JScript">
</SCRIPT>

VBSc...
<SCRIPT LANGUAGE="VBScript">
Function DropUp()
End Function
</SCRIPT>

Visual
Data...
Procedure OnComDropUp
Forward Send OnComDropUp

End_Procedure

Visual
Objects

METHOD OCX_DropUp() CLASS MainDialog
RETURN NIL

X++

```
void onEvent_DropUp()
{
}
```

XBasic

```
function DropUp as v ()
end function
```

dBASE

```
function nativeObject_DropUp()
return
```

event EditChange (ColIndex as Long)

Fired when the user has taken an action that may have altered text in an edit control.

Type	Description
ColIndex as Long	A long expression that indicates the index of the edit control that's altered. If -1 it indicates that the FilterFor field of the control has been changed.

The EditChange event notifies your application that the user alters the text of one of the control's edit labels. The control supports single or multiple edit controls in the label area. Use the [SingleEdit](#) property to specify that a single edit control is used. Use the [EditText](#) property to determine the text of the edit control. Use the [KeyDown](#) event to notify your application that the user presses a key. Use the [KeyUp](#) event to notify your application that the user releases a key. If the control's [Style](#) property is DropDownList, the EditChange event is never fired. Use the [AutoComplete](#) property to allow typing values that are not in the column. Use the [SelectionChanged](#) event to notify your application that the user changes the selected item.

The following VB sample prints the text while editing:

```
Private Sub ComboBox1_EditChange(ByVal ColIndex As Long)
    Debug.Print ComboBox1.EditText(ColIndex)
End Sub
```

The following C++ sample prints the text while editing:

```
void OnEditChangeCombobox1(long ColIndex)
{
    OutputDebugString( m_combobox.GetEditText( COleVariant( ColIndex ) ) );
}
```

The following VB.NET sample prints the text while editing:

```
Private Sub AxComboBox1_EditChange(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_EditChangeEvent) Handles
AxComboBox1.EditChange
    With AxComboBox1
        Debug.WriteLine(.get_EditText(e.colIndex))
    End With
End Sub
```

The following C# sample prints the text while editing:

```
private void axComboBox1_EditChange(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_EditChangeEvent e)
{
    System.Diagnostics.Debug.WriteLine(axComboBox1.get_EditText(e.colIndex));
}
```

The following VFP sample prints the text while editing:

```
*** ActiveX Control Event ***
LPARAMETERS colindex

with thisform.ComboBox1
    wait window nowait .EditText(colindex)
endwith
```

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

```
C# private void EditChange(object sender,int ColIndex)
{
}
```

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

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

```
C# private void EditChange(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_EditChangeEvent e)
{
}
```

```
C++ void OnEditChange(long ColIndex)
{
}
```

C++
Builder

```
void __fastcall EditChange(TObject *Sender,long ColIndex)
{
}
```

Delphi

```
procedure EditChange(ASender: TObject; ColIndex : Integer);
begin
end;
```

Delphi 8
(.NET
only)

```
procedure EditChange(sender: System.Object; e:
AxEXCOMBOBOXLib._IComboBoxEvents_EditChangeEvent);
begin
end;
```

Powe...

```
begin event EditChange(long ColIndex)
end event EditChange
```

VB.NET

```
Private Sub EditChange(ByVal sender As System.Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_EditChangeEvent) Handles EditChange
End Sub
```

VB6

```
Private Sub EditChange(ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub EditChange(ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS ColIndex
```

Xbas...

```
PROCEDURE OnEditChange(oComboBox,ColIndex)
RETURN
```

Syntax for EditChange event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="EditChange(ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function EditChange(ColIndex)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComEditChange Integer IIColIndex  
Forward Send OnComEditChange IIColIndex  
End_Procedure
```

```
Visual  
Objects METHOD OCX_EditChange(ColIndex) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_EditChange(int _ColIndex)  
{  
}
```

```
XBasic function EditChange as v (ColIndex as N)  
end function
```

```
dBASE function nativeObject_EditChange(ColIndex)  
return
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Use the 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 extree1.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) */
        extree1.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 ( !extree1.Items().EnableItem( extree1.EventParam( 2 /*NewItem*/ ) ) )
            extree1.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, AxEXTREELib._IComboBoxEvents_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:
AxEXTREELib._IComboBoxEvents_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  
AxEXTREELib._IComboBoxEvents_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(oComboBox,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 IEventID  
Forward Send OnComEvent IEventID  
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 the filter was changed. */*not supported in the lite version*/*

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

Use the FilterChange event to notify your application that the control's filter is changed. 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. Use the [FilterBarCaption](#) property to change the caption of the control's filter bar.

The following VB sample changes the caption in the control's filter bar when the user changes the filter:

```
Private Sub ComboBox1_FilterChange()  
    With ComboBox1  
        .FilterBarCaption = "custom filter caption"  
    End With  
End Sub
```

The following C++ sample changes the caption in the control's filter bar when the user changes the filter:

```
void OnFilterChangeCombobox1()  
{  
    m_combobox.SetFilterBarCaption( "custom filter caption" );  
}
```

The following VB.NET sample changes the caption in the control's filter bar when the user changes the filter:

```
Private Sub AxComboBox1_FilterChange(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxComboBox1.FilterChange  
    With AxComboBox1  
        .FilterBarCaption = "custom filter caption"  
    End With  
End Sub
```

The following C# sample changes the caption in the control's filter bar when the user changes the filter:

```
private void axComboBox1_FilterChange(object sender, EventArgs e)
{
    axComboBox1.FilterBarCaption = "custom filter caption";
}
```

The following VFP sample changes the caption in the control's filter bar when the user changes the filter:

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

with thisform.ComboBox1
    .FilterBarCaption = "custom filter caption"
endwith
```

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;
```

Power...

```
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(oComboBox)  
RETURN
```

Syntax for FilterChange event, **ICOM** 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 FormatColumn (Item as HITEM, ColIndex as Long, Value as Variant)

Fired when a cell requires to format its caption.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being formatted.
ColIndex as Long	A long expression that indicates the index of the column being formatted.
Value as Variant	A Variant value that indicates the value being displayed in the cell. By default, the Value parameter is initialized with the CellCaption property.

Use the FormatColumn event to display a string different than the CellCaption 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 \$50 instead 50. Also, you can use the FormatColumn event to display item's index in the column, or to display the result of some operations based on the cells in the item (totals, currency conversion and so on).

The following VB samples use the FormatCurrency function, to display a number as a currency. The FormatCurrency VB function returns an expression formatted as a currency value using the currency symbol defined in the system control panel.

```
ComboBox1.Columns("Freight").FireFormatColumn = True  
ComboBox1.Columns("Freight").HeaderBold = True  
ComboBox1.Columns("Freight").Alignment = RightAlignment
```

```
Private Sub ComboBox1_FormatColumn(ByVal Item As  
EXCOMBOBOXLibCtl.HITEM, ByVal ColIndex As Long, Value As Variant)
```

```
Value = FormatCurrency(Value, 2) ' The FormatCurrency is a VB function
```

```
End Sub
```

if the sample looks like following:

Freight
\$12.75
\$10.19
\$52.84
\$0.59
\$8.56
\$42.11
\$15.51
\$108.26
\$84.21

Freight
12.75
10.19
52.84
0.59
8.56
42.11
15.51
100.26
84.21

```

ComboBox1.Columns("Freight").FireFormatColumn = False
ComboBox1.Columns("Freight").HeaderBold = True
ComboBox1.Columns("Freight").Alignment = RightAlignment

```

For instance, you can use the FormatColumn event to display "Yes" or "No" caption for a boolean column. The following VB sample shows how to do it:

```

Private Sub ComboBox1_FormatColumn(ByVal Item As EXCOMBOBOXLibCtl.HITEM, ByVal
ColIndex As Long, Value As Variant)
    Value = If(Value < 50, "Yes", "No")
End Sub

```

The following VB sample displays the result of adding (concatenating) of two cells:

```

Private Sub ComboBox1_FormatColumn(ByVal Item As EXCOMBOBOXLibCtl.HITEM, ByVal
ColIndex As Long, Value As Variant)
    With ComboBox1.Items
        Value = .CellCaption(Item, 0) + .CellCaption(Item, 1)
    End With
End Sub

```

The following C++ sample displays a date column using a format like "Saturday, January 31, 2004":

```

void OnFormatColumnComboBox1(long Item, 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 a date column using LongDate format:

```
Private Sub AxComboBox1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_FormatColumnEvent) Handles
AxComboBox1.FormatColumn
    e.value = DateTime.Parse(e.value).ToLongDateString()
End Sub
```

The following C# sample displays a date column using LongDate format:

```
private void axComboBox1_FormatColumn(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_FormatColumnEvent e)
{
    e.value = DateTime.Parse(e.value.ToString()).ToLongDateString();
}
```

The following VFP sample displays the item's index using the FormatColumn event:

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

with thisform.ComboBox1.Items
    .DefaultItem = item
    value = .ItemToIndex(0)
endwith
```

before running the sample please make sure that the :

```
application.AutoYield = .f.
```

is called during the Form.Init event.

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

```
C# private void FormatColumn(object sender,int Item,int ColIndex,ref object Value)
{
}
```

```
VB Private Sub FormatColumn(ByVal sender As System.Object,ByVal Item 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,
AxEXCOMBOBOXLib._IComboBoxEvents_FormatColumnEvent e)
{
}
```

```
C++ void OnFormatColumn(long Item,long ColIndex,VARIANT FAR* Value)
{
}
```

```
C++ Builder void __fastcall FormatColumn(TObject *Sender,Excomboboxlib_tlb::HITEM
Item,long ColIndex,Variant * Value)
{
}
```

```
Delphi procedure FormatColumn(ASender: TObject; Item : HITEM;ColIndex : Integer;var
Value : OleVariant);
begin
end;
```

```
Delphi 8 (.NET only) procedure FormatColumn(sender: System.Object; e:
AxEXCOMBOBOXLib._IComboBoxEvents_FormatColumnEvent);
begin
end;
```

```
Powe... begin event FormatColumn(long Item,long ColIndex,any Value)
end event FormatColumn
```

```
VB.NET Private Sub FormatColumn(ByVal sender As System.Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_FormatColumnEvent) Handles
FormatColumn
End Sub
```

```
VB6 Private Sub FormatColumn(ByVal Item As EXCOMBOBOXLibCtl.HITEM,ByVal
ColIndex As Long,Value As Variant)
End Sub
```

VBA

```
Private Sub FormatColumn(ByVal Item As Long,ByVal ColIndex As Long,Value As Variant)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,Value
```

Xbas...

```
PROCEDURE OnFormatColumn(oComboBox,Item,ColIndex,Value)
RETURN
```

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

Java...

```
<SCRIPT EVENT="FormatColumn(Item,ColIndex,Value)" LANGUAGE="JScript">
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">
Function FormatColumn(Item,ColIndex,Value)
End Function
</SCRIPT>
```

Visual
Data...

```
Procedure OnComFormatColumn HITEM IItem Integer IColIndex Variant IValue
Forward Send OnComFormatColumn IItem IColIndex IValue
End_Procedure
```

Visual
Objects

```
METHOD OCX_FormatColumn(Item,ColIndex,Value) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_FormatColumn(int _Item,int _ColIndex,COMVariant /*variant*/
_Value)
{
}
```

XBasic

```
function FormatColumn as v (Item as OLE::Exontrol.ComboBox.1::HITEM,ColIndex
as N,Value as A)
end function
```

dBASE

```
function nativeObject_FormatColumn(Item,ColIndex,Value)
return
```


event InsertItem (Item as HITEM)

Occurs after a new Item has been inserted to Items collection.

Type	Description
Item as HITEM	A long expression that indicates the handle of the newly inserted item.

The InsertItem event notifies your application that a new item is inserted. Use the InsertItem event to associate extra data to the item. The InsertItem event is fired when [PutItems](#), [AddItem](#) or [InsertItem](#) property of the Items object is called. Use the [Add](#) method to add new columns to Columns Collection. Use the [Def](#) property to specify a common property for all cells in the same 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 and background colors for cells in the first column, as soon as they are inserted:

```
Private Sub ComboBox1_InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)
    With ComboBox1.Items
        .CellBackColor(Item, 0) = vbBlue
        .CellForeColor(Item, 0) = vbWhite
    End With
End Sub
```

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

```
With ComboBox1.Columns(0)
    .Def(exCellBackColor) = RGB(240, 240, 240)
End With
```

The following C++ sample changes the foreground and background colors for cells in the first column, as soon as they are inserted:

```
#include "Items.h"
void OnInsertItemCombobox1(long Item)
{
    if ( IsWindow( m_combobox.m_hWnd ) )
    {
        CItems items = m_combobox.GetItems();
```

```

COleVariant vtItem( Item ), vtColumn( long(0) );
items.SetCellBackColor( vtItem, vtColumn, RGB(0,0,255) );
items.SetCellForeColor( vtItem, vtColumn, RGB(255,255,255) );
}
}

```

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

```

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

```

The following VB.NET sample changes the foreground and background colors for cells in the first column, as soon as they are inserted:

```

Private Sub AxComboBox1_InsertItem(ByVal sender As System.Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_InsertItemEvent) Handles
AxComboBox1.InsertItem
    With AxComboBox1.Items
        .CellBackColor(e.item, 0) = ToUInt32(Color.Blue)
        .CellForeColor(e.item, 0) = ToUInt32(Color.White)
    End With
End Sub

```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expression,

```

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

```

With AxComboBox1.Columns(0)
    .Def(EXCOMBOBOXLib.DefColumnEnum.exCellBackColor) =
ToUInt32(Color.WhiteSmoke)

```

The following C# sample changes the foreground and background colors for cells in the first column, as soon as they are inserted:

```
private void axComboBox1_InsertItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_InsertItemEvent e)
{
    EXCOMBOBOXLib.Items items = axComboBox1.Items;
    items.set_CellBackColor(e.item, 0, ToUInt32(Color.Blue));
    items.set_CellForeColor(e.item, 0, ToUInt32(Color.White));
}
```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expression,

```
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 all cells in the first column:

```
axComboBox1.Columns[0].set_Def(EXCOMBOBOXLib.DefColumnEnum.exCellBackColor,
ToUInt32(Color.WhiteSmoke));
```

The following VFP sample changes the foreground and background colors for cells in the first column, as soon as they are inserted:

```
*** ActiveX Control Event ***
LPARAMETERS item

with thisform.ComboBox1.Items
    .DefaultItem = item
    .CellBackColor(0,0) = RGB(0,0,255)
    .CellForeColor(0,0) = RGB(255,255,255)
```

```
endwith
```

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

```
with thisform.ComboBox1.Columns(0)
    .Def( 4 ) = RGB(240, 240, 240)
endwith
```

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

```
C# private void InsertItem(object sender,int Item)
{
}
```

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

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

```
C# private void InsertItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_InsertItemEvent e)
{
}
```

```
C++ void OnInsertItem(long Item)
{
}
```

```
C++ Builder void __fastcall InsertItem(TObject *Sender,Excomboboxlib_tlb::HITEM Item)
{
}
```

```
Delphi procedure InsertItem(ASender: TObject; Item : HITEM);
begin
end;
```

```
Delphi 8 (.NET only) procedure InsertItem(sender: System.Object; e:
AxEXCOMBOBOXLib._IComboBoxEvents_InsertItemEvent);
```

```
begin  
end;
```

```
Powe... begin event InsertItem(long Item)  
end event InsertItem
```

```
VB.NET Private Sub InsertItem(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_InsertItemEvent) Handles InsertItem  
End Sub
```

```
VB6 Private Sub InsertItem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)  
End Sub
```

```
VBA Private Sub InsertItem(ByVal Item As Long)  
End Sub
```

```
VFP LPARAMETERS Item
```

```
Xbas... PROCEDURE OnInsertItem(oComboBox,Item)  
RETURN
```

Syntax for InsertItem event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="InsertItem(Item)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function InsertItem(Item)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComInsertItem HITEM lItem  
Forward Send OnComInsertItem lItem  
End_Procedure
```

```
Visual  
Objects METHOD OCX_InsertItem(Item) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_InsertItem(int _Item)
```

```
{  
}
```

XBasic

```
function InsertItem as v (Item as OLE::Exontrol.ComboBox.1::HITEM)  
end function
```

dBASE

```
function nativeObject_InsertItem(Item)  
return
```

event KeyDown (KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use `KeyDown` and [KeyUp](#) event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the `And` operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
```

In a procedure, you can test for any combination of conditions, as in this example:
If `AltDown And CtrlDown` Then

The [EditChange](#) event notifies your application that the user alters the control's edit. Use the [SelectionChanged](#) event to notify your application that the user changes the selected item. Use the [EditText](#) property to determine the text of the edit control. Use the [DropDown](#) method to show programmatically the control's drop down portion. By default, the control shows the drop down portion of the control when the user presses the F4 key. Use the [Key](#) property to replace the default action when a certain key is pressed.

The following VB sample shows the control's drop down portion when the user presses the F3 key:

```
Private Sub ComboBox1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF3) Then
        With ComboBox1
```

```
.DropDown() = True
End With
End If
End Sub
```

The following C++ sample shows the control's drop down portion when the user presses the F3 key:

```
void OnKeyDownCombobox1(short FAR* KeyCode, short Shift)
{
    if ( *KeyCode == VK_F3 )
    {
        COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
        m_combobox.SetDropDown( vtMissing, TRUE );
    }
}
```

The following VB.NET sample shows the control's drop down portion when the user presses the F3 key:

```
Private Sub AxComboBox1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_KeyDownEvent) Handles
AxComboBox1.KeyDownEvent
    If (Convert.ToUInt32(e.keyCode) = Convert.ToUInt32(Keys.F3)) Then
        With AxComboBox1
            .set_DropDown(True)
        End With
    End If
End Sub
```

The following C# sample shows the control's drop down portion when the user presses the F3 key:

```
private void axComboBox1_KeyDownEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_KeyDownEvent e)
{
    if (Convert.ToUInt32(e.keyCode) == Convert.ToUInt32(Keys.F3))
        axComboBox1.set_DropDown(true);
}
```

The following VFP sample shows the control's drop down portion when the user presses the F3 key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 114 ) && F3
    with thisform.ComboBox1
        .Object.DropDown("") = .t.
    endwhile
endif
```

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,
AxEXCOMBOBOXLib._IComboBoxEvents_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:  
AxEXCOMBOBOXLib._IComboBoxEvents_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  
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,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. The [EditChange](#) event notifies your application that the user alters the control's edit. Use the [SelectionChanged](#) event to notify your application that the user changes the selected item. Use the [EditText](#) property to determine the text of the edit control.

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,
AxEXCOMBOBOXLib._IComboBoxEvents_KeyPressEvent e)
{
}
```

```
C++ void OnKeyPress(short FAR* KeyAscii)
{
}
```

```
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:
AxEXCOMBOBOXLib._IComboBoxEvents_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
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,KeyAscii)
RETURN
```

Syntax for KeyPress event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function KeyPress(KeyAscii)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComKeyPress Short Integer KeyAscii  
Forward Send OnComKeyPress Integer KeyAscii  
End_Procedure
```

```
Visual  
Objects METHOD OCX_KeyPress(KeyAscii) CLASS MainDialog  
RETURN NIL
```

```
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,
AxEXCOMBOBOXLib._IComboBoxEvents_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:
AxEXCOMBOBOXLib._IComboBoxEvents_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
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,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 IIKeyCode Short IIShift  
    Forward Send OnComKeyUp IIKeyCode IIShift  
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 `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;

Power... 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(oComboBox)
RETURN

Syntax for LayoutChanged event, **ICOM** 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

```
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 [MouseDown](#) event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the [ItemFromPoint](#) property to determine the cell or the item over the cursor. Use the [CloseOnDbClick](#) property to specify whether the drop down portion of the control is closed when user clicks or double clicks the list. Use the [Change](#) event to notify your application that user clicks an item or presses the ENTER key. Use the [SelectionChange](#) event to notify your application that a new item is selected.

The following VB sample displays the cell from the cursor:

```
Private Sub ComboBox1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        Dim i As Long, c As Long, hit As EXCOMBOBOXLibCtl.HitTestInfoEnum
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If Not (i = 0) Then
            With .Items
                Debug.Print .CellCaption(i, c)
            End With
        End If
    End Sub
```

```
End With
End Sub
```

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

```
void OnMouseDownCombobox1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, i = m_combobox.GetItemFromPoint( X, Y, &c, &hit );
    if ( i != 0 )
        OutputDebugString( V2S( &m_combobox.GetItems().GetCellCaption(COleVariant(i),
COleVariant(c) ) ) );
}
```

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

```
Private Sub AxComboBox1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_MouseDownEvent) Handles
AxComboBox1.MouseDownEvent
    With AxComboBox1
        Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (i >= 0) Then
            Debug.WriteLine(.Items.CellCaption(i, c))
        End If
    End With
End Sub
```

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

```
private void axComboBox1_MouseDownEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_MouseDownEvent e)
{
    EXCOMBOBOXLib.HitTestInfoEnum h;
    int c = 0, i = axComboBox1.get_ItemFromPoint(e.x, e.y, out c, out h);
    if (i != 0)
        System.Diagnostics.Debug.WriteLine(axComboBox1.Items.get_CellCaption(i,
c).ToString());
}
```

The following VFP sample displays the cell from the cursor:

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

```
LPARAMETERS button, shift, x, y
```

```
With thisform.ComboBox1
```

```
    local c, hit
```

```
    c = 0
```

```
    hit = 0
```

```
    .Items.DefaultItem = .ItemFromPoint(X , Y, @c, @hit)
```

```
    If (.Items.DefaultItem >= 0) Then
```

```
        wait window nowait .Items.CellCaption(0, c)
```

```
    EndIf
```

```
EndWith
```

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,
AxEXCOMBOBOXLib._IComboBoxEvents_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: AxEXCOMBOBOXLib._IComboBoxEvents_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 AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,Button,Shift,X,Y)  
RETURN
```

Syntax for MouseDown event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="MouseDown(Button,Shift,X,Y)" LANGUAGE="JScript">  
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">  
Function MouseDown(Button,Shift,X,Y)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComMouseDown Short IButton Short IShift OLE_XPOS_PIXELS IIX  
OLE_YPOS_PIXELS IY  
    Forward Send OnComMouseDown IButton IShift IIX IY  
End_Procedure
```

Visual
Objects

```
METHOD OCX_MouseDown(Button,Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_MouseDown(int _Button,int _Shift,int _X,int _Y)  
{  
}
```

XBasic

```
function MouseDown as v (Button as N,Shift as N,X as  
OLE::Exontrol.ComboBox.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.ComboBox.1::OLE_YPOS_PIXELS)  
end function
```

dBASE

```
function nativeObject_MouseDown(Button,Shift,X,Y)  
return
```

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 get the item from cursor. Use the [ColumnFromPoint](#) property to get the column from point. Use the [Background\(exHoverColumn\)](#) property to change the visual appearance of the column's header when the cursor hovers it. Use the [AnchorFromPoint](#) property to retrieve the identifier of the anchor from the point.

The following VB sample prints the cell from the caption and the hit test code:

```
Private Sub ComboBox1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With ComboBox1
        Dim i As Long, c As Long, hit As EXCOMBOBOXLibCtl.HitTestInfoEnum
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If Not (i = 0) Then
            With .Items
                Debug.Print .CellCaption(i, c) & " HT = " & hit
            End With
        End If
    End Sub
```

```
End With
End Sub
```

The following C++ sample prints the cell from the caption and the hit test code:

```
void OnMouseMoveCombobox1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, i = m_combobox.GetItemFromPoint( X, Y, &c, &hit );
    if ( i != 0 )
    {
        CString strOutput = V2S( &m_combobox.GetItems().GetCellCaption(COleVariant(i),
COleVariant(c) ) );
        strOutput += " HT = ";
        strOutput += V2S(COleVariant(hit));
        strOutput += "\r\n";
        OutputDebugString( strOutput );
    }
}
```

The following VB.NET sample prints the cell from the caption and the hit test code:

```
Private Sub AxComboBox1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib.IComboBoxEvents_MouseMoveEvent) Handles
AxComboBox1.MouseMoveEvent
    With AxComboBox1
        Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (i >= 0) Then
            Debug.WriteLine(.Items.CellCaption(i, c) & " HT = " & hit.ToString())
        End If
    End With
End Sub
```

The following C# sample prints the cell from the caption and the hit test code:

```
private void axComboBox1_MouseMoveEvent(object sender,
AxEXCOMBOBOXLib.IComboBoxEvents_MouseMoveEvent e)
{
    EXCOMBOBOXLib.HitTestInfoEnum hit;
```

```

int c = 0, i = axComboBox1.GetItemFromPoint(e.x, e.y, out c, out hit);
if (i != 0)
    System.Diagnostics.Debug.WriteLine(axComboBox1.Items.getCellCaption(i,
c).ToString() + " HT = " + hit.ToString() );
}

```

The following VFP sample prints the cell from the caption and the hit test code:

```

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

With thisform.ComboBox1
    local c, hit
    c = 0
    hit = 0
    .Items.DefaultItem = .ItemFromPoint(X , Y, @c, @hit)
    If (.Items.DefaultItem >= 0) Then
        wait window nowait .Items.CellCaption(0, c) + " HT = " + str(hit)
    EndIf
EndWith

```

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,
AxEXCOMBOBOXLib._IComboBoxEvents_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: AxEXCOMBOBOXLib._IComboBoxEvents_MouseMoveEvent);
begin
end;
```

Power...

```
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 AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,Button,Shift,X,Y)
```

Syntax for MouseMove event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="MouseMove(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">
Function MouseMove(Button,Shift,X,Y)
End Function
</SCRIPT>
```

```
Visual Data... Procedure OnComMouseMove Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
    Forward Send OnComMouseMove IButton IShift IIX IY
End_Procedure
```

```
Visual Objects METHOD OCX_MouseMove(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

```
X++ void onEvent_MouseMove(int _Button,int _Shift,int _X,int _Y)
{
}
```

```
XBasic function MouseMove as v (Button as N,Shift as N,X as
OLE::Exontrol.ComboBox.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.ComboBox.1::OLE_YPOS_PIXELS)
end function
```

```
dBASE function nativeObject_MouseMove(Button,Shift,X,Y)
return
```

event MouseUp (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user releases a mouse button.

| Type | Description |
|----------------------|--|
| Button as Integer | An integer that identifies the button that was pressed to cause the event. |
| Shift as Integer | An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released. |
| X as OLE_XPOS_PIXELS | A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates. |
| Y as OLE_YPOS_PIXELS | A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates. |

Use a [MouseDown](#) or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the [ItemFromPoint](#) property to determine the cell or the item over the cursor. Use the [Change](#) event to notify your application that user clicks an item or presses the ENTER key. Use the [SelectionChange](#) event to notify your application that a new item is selected. Use the [CellImages](#) property to assign multiple icons to a cell.

The following VB sample displays the index of the icon being clicked in a cell:

```
Private Sub ComboBox1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim i As HITEM, h As HitTestInfoEnum, c As Long
    With ComboBox1
        i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
    End With
    If (i <> 0) Or (c <> 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 displays the index of the icon being clicked in a cell:

```
void OnMouseUpCombobox1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, i = m_combobox.GetItemFromPoint( X, Y, &c, &hit );
    if ( i != 0 )
        if ( 68 /*exHTCellIcon*/ == (hit & 68 /*exHTCellIcon*/ ) )
        {
            CString strOutput = V2S( &m_combobox.GetItems().GetCellCaption(COleVariant(i),
COleVariant(c) ) );
            strOutput += " Index = ";
            strOutput += V2S(COleVariant(hit >> 16));
            strOutput += "\r\n";
            OutputDebugString( strOutput );
        }
}
```

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

```
Private Sub AxComboBox1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib.IComboBoxEvents_MouseUpEvent) Handles
AxComboBox1.MouseUpEvent
    With AxComboBox1
        Dim i As Integer, c As Integer, hit As EXCOMBOBOXLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (i >= 0) Then
            If (hit And EXCOMBOBOXLib.HitTestInfoEnum.exHTCellIcon) =
EXCOMBOBOXLib.HitTestInfoEnum.exHTCellIcon Then
                Debug.WriteLine(.Items.CellCaption(i, c) & " Index = " & ((hit And &HFFFF0000)
/ 65536))
            End If
        End If
    End With
End Sub
```

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

```
private void axComboBox1_MouseUpEvent(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_MouseUpEvent e)
{
    EXCOMBOBOXLib.HitTestInfoEnum hit;
    int c = 0, i = axComboBox1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i != 0)
        System.Diagnostics.Debug.WriteLine(axComboBox1.Items.get_CellCaption(i,
c).ToString() + " Index = " + (Convert.ToUInt32(hit) >> 16).ToString());
}
```

The following VFP sample displays the cell from the cursor:

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

local c, hit
c = 0
hit = 0
with thisform.ComboBox1
    .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
    if ( .Items.DefaultItem <> 0 )
        if ( bitand( hit, 68 )= 68 )
            wait window nowait .Items.CellCaption( 0, c ) + " Index=" + Str( Int((hit - 68)/65536) )
        endif
    endif
endwith
```

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,
AxEXCOMBOBOXLib._IComboBoxEvents_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:
AxEXCOMBOBOXLib._IComboBoxEvents_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
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,Button,Shift,X,Y)
RETURN

Syntax for MouseUp event, **ICOM** version (others), on:

Java... <SCRIPT EVENT="MouseUp(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>

VBS... <SCRIPT LANGUAGE="VBScript">
Function MouseUp(Button,Shift,X,Y)
End Function
</SCRIPT>

Visual Data... Procedure OnComMouseUp Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
Forward Send OnComMouseUp IButton IShift IIX IY
End_Procedure

Visual Objects METHOD OCX_MouseUp(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL

X++ void onEvent_MouseUp(int _Button,int _Shift,int _X,int _Y)
{
}

XBasic function MouseUp as v (Button as N,Shift as N,X as
OLE::Exontrol.ComboBox.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.ComboBox.1::OLE_YPOS_PIXELS)
end function

dBASE function nativeObject_MouseUp(Button,Shift,X,Y)
return

event NotInList ()

Occurs when the user enters a value in the text box portion of a combo box that isn't in the combo box list.

Type

Description

The NotInList event notifies your application that the user enters a value in the text box portion of the combo box (label) that isn't in the combo box list. The NotInList property is fired when user presses the enter key or the control loses the focus. The NotInList event is fired only if the [FireNotInList](#) property is True. The NotInList event is not fired if the [Style](#) property is DropDownList. The NotInList event may **not** be fired if the [AutoComplete](#) property is True. This way the control's label portion always display the selection in the control's list as user types characters in the control's label area, so the text box portion of the control can't have a value that it is not in the combo box list. Use the [EditText](#) property to get the text from the control's label. Use the [SelectItem](#) property to select an item. Use the [AddItem](#) or [InsertItem](#) method to insert a new item to the control's drop down list.

Before running any of the following samples please make sure that the Style property is DropDown, FireNotInList property is True, AutoComplete property is False.

The following VB sample adds a new item, if it is not in the combo box list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
Private Sub ComboBox1_NotInList()  
    With ComboBox1  
        Dim h As EXCOMBOBOXLibCtl.HITEM  
        h = .Items.AddItem(.EditText(0))  
        .Items.SelectItem(h) = True  
    End With  
End Sub
```

```
Private Sub Form_Load()  
    With ComboBox1  
        .BeginUpdate  
        .AutoComplete = False  
        .FireNotInList = True  
        .ColumnAutoResize = True  
        .HeaderVisible = False  
        .Columns.Add "Column"  
    End With  
End Sub
```

```
.AddItem "Ana"  
.AddItem "Maria"  
End With  
.EndUpdate  
End With  
End Sub
```

The following JScript sample adds a new item if the control's drop down list doesn't contain it :

```
<SCRIPT FOR="ComboBox1" EVENT="NotInList()" LANGUAGE="javascript">  
  
obj = document.getElementById ( "ComboBox1" );  
h = obj.Items.AddItem(obj.EditText(0));  
obj.Items.SelectItem(h) = true  
  
</SCRIPT>
```

The following C++ sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
void OnNotInListCombobox1()  
{  
    CItems items = m_combobox.GetItems();  
    long hItem = items.AddItem( COleVariant(  
m_combobox.GetEditText(COleVariant(long(0))) ) );  
    items.SetSelectItem( hItem, TRUE );  
}
```

The following VB.NET sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
Private Sub AxComboBox1_NotInList(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxComboBox1.NotInList  
    With AxComboBox1  
        Dim h As Integer = .Items.AddItem(.get_EditText(0))  
        .Items.SelectItem(h) = True  
    End With  
End Sub
```

The following C# sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

```
private void axComboBox1_NotInList(object sender, EventArgs e)
{
    EXCOMBOBOXLib.Items items = axComboBox1.Items;
    int hItem = items.AddItem(axComboBox1.get_EditText(0));
    items.set_SelectItem(hItem, true);
}
```

The following VFP sample adds a new entry to the drop down list, when the user presses the ENTER key or the control loses the focus, using the NotInList event:

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

```
With thisform.ComboBox1
    .Items.DefaultItem = .Items.AddItem(.EditText(0))
    .Items.SelectItem(0) = .t.
EndWith
```

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

```
C# private void NotInList(object sender)
{
}
```

```
VB Private Sub NotInList(ByVal sender As System.Object) Handles NotInList
End Sub
```

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

```
C# private void NotInList(object sender, EventArgs e)
{
}
```

```
C++ void OnNotInList()
{
}
```

C++
Builder

```
void __fastcall NotInList(TObject *Sender)
{
}
```

Delphi

```
procedure NotInList(ASender: TObject; );
begin
end;
```

Delphi 8
(.NET
only)

```
procedure NotInList(sender: System.Object; e: System.EventArgs);
begin
end;
```

Power...

```
begin event NotInList()
end event NotInList
```

VB.NET

```
Private Sub NotInList(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles NotInList
End Sub
```

VB6

```
Private Sub NotInList()
End Sub
```

VBA

```
Private Sub NotInList()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnNotInList(oComboBox)
RETURN
```

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

Java...

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

VBS...

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

Visual
Data...

```
Procedure OnComNotInList  
    Forward Send OnComNotInList  
End_Procedure
```

Visual
Objects

```
METHOD OCX_NotInList() CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_NotInList()  
{  
}
```

XBasic

```
function NotInList as v ()  
end function
```

dBASE

```
function nativeObject_NotInList()  
return
```

event OffsetChanged (Horizontal as Boolean, NewVal as Long)

Occurs when the scroll position has been changed.

| Type | Description |
|-----------------------|---|
| Horizontal as Boolean | A boolean expression that indicates whether the horizontal or 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 bars to the control when required. The `OffsetChanged` event notifies your application that user changes the position of one of the control's scroll bars. The [OversizeChanged](#) event notifies your application that the range of control's scroll bar is changed.

The following VB sample displays the new scroll position when user scrolls horizontally the control:

```
Private Sub ComboBox1_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 VC sample displays the new scroll position when the user scrolls vertically the control:

```
void OnOffsetChangedCombobox1(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 AxComboBox1_OffsetChanged(ByVal sender As Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_OffsetChangedEvent) Handles
AxComboBox1.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 axComboBox1_OffsetChanged(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_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
```

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,
AxEXCOMBOBOXLib._IComboBoxEvents_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:
AxEXCOMBOBOXLib._IComboBoxEvents_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
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,Horizontal,NewVal)  
RETURN
```

Syntax for OffsetChanged event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="OffsetChanged(Horizontal,NewVal)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function OffsetChanged(Horizontal,NewVal)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComOffsetChanged Boolean IIHorizontal Integer IINewVal  
Forward Send OnComOffsetChanged IIHorizontal IINewVal  
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
```

event **OversizeChanged** (Horizontal as Boolean, NewVal as Long)

Occurs when the right range of the scroll has been changed.

| Type | Description |
|-----------------------|--|
| Horizontal as Boolean | A boolean expression that indicates whether the range of the horizontal or 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 **OversizeChanged** event is not fired. The **OversizeChanged** event notifies your application that the range of the control's scroll bars is changed. The [OffsetChanged](#) event property notifies your application that the position of the control's scroll bars is changed.

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,
AxEXCOMBOBOXLib._IComboBoxEvents_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:  
AxEXCOMBOBOXLib._IComboBoxEvents_OversizeChangedEvent);  
begin  
end;
```

Power...

```
begin event OversizeChanged(boolean Horizontal,long NewVal)  
end event OversizeChanged
```

VB.NET

```
Private Sub OversizeChanged(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_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(oComboBox,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 IIHorizontal Integer IINewVal  
Forward Send OnComOversizeChanged IIHorizontal IINewVal  
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

The RClick event notifies your application that user right clicks the drop down portion of the control. Use the [MouseDown](#) or [MouseUp](#) event if you require the cursor coordinates. The [Style](#) property specifies the style of the control. Use the [Click](#) event to notify your application that the user clicks an item. Use the [Change](#) event to notify your application that the user presses the ENTER key. Use the [SelectionChanged](#) event to notify whether the user changes the selection.

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;
```

Power...
begin event RClick()
end event RClick

VB.NET
Private Sub RClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RClick
End Sub

VB6
Private Sub RClick()
End Sub

VBA
Private Sub RClick()
End Sub

VFP
LPARAMETERS nop

Xbas...
PROCEDURE OnRClick(oComboBox)
RETURN

Syntax for RClick event, **ICOM** version (others), on:

Java...
<SCRIPT EVENT="RClick()" LANGUAGE="JScript">
</SCRIPT>

VBSc...
<SCRIPT LANGUAGE="VBScript">
Function RClick()
End Function
</SCRIPT>

Visual
Data...
Procedure OnComRClick
Forward Send OnComRClick

End_Procedure

Visual
Objects

METHOD OCX_RClick() CLASS MainDialog
RETURN NIL

X++

```
void onEvent_RClick()
{
}
```

XBasic

```
function RClick as v ()
end function
```

dBASE

```
function nativeObject_RClick()
return
```

event RemoveColumn (Column as Column)

Fired before removing a Column.

| Type | Description |
|----------------------------------|--------------------------------|
| Column as Column | A Column object being removed. |

The RemoveColumn event notifies your application that a column is removed. Use the [Remove](#) method to remove programmatically a column. Use the RemoveColumn event to release any extra data associated to a column. Use the [Data](#) property to assign an extra data to a column. Use the [Clear](#) method to clear the columns collection. The control fires the [RemoveItem](#) when the user removes an item. Use the [RemoveAllItems](#) method to remove all items.

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

```
C# private void RemoveColumn(object sender,exontrol.EXCOMBOBOXLib.Column
Column)
{
}
```

```
VB Private Sub RemoveColumn(ByVal sender As System.Object,ByVal Column As
exontrol.EXCOMBOBOXLib.Column) Handles RemoveColumn
End Sub
```

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

```
C# private void RemoveColumn(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_RemoveColumnEvent e)
{
}
```

```
C++ void OnRemoveColumn(LPDISPATCH Column)
{
}
```

```
C++ Builder void __fastcall RemoveColumn(TObject *Sender,Excomboboxlib_tlb::IColumn
*Column)
{
}
```

```
Delphi procedure RemoveColumn(ASender: TObject; Column : IColumn);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure RemoveColumn(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_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  
AxEXCOMBOBOXLib._IComboBoxEvents_RemoveColumnEvent) Handles  
RemoveColumn  
End Sub
```

```
VB6 Private Sub RemoveColumn(ByVal Column As EXCOMBOBOXLibCtl.IColumn)  
End Sub
```

```
VBA Private Sub RemoveColumn(ByVal Column As Object)  
End Sub
```

```
VFP LPARAMETERS Column
```

```
Xbas... PROCEDURE OnRemoveColumn(oComboBox,Column)  
RETURN
```

Syntax for RemoveColumn event, **ICOM** 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 IIColumn  
    Forward Send OnComRemoveColumn IIColumn  
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.ComboBox.1::IIColumn)  
end function
```

dBASE

```
function nativeObject_RemoveColumn(Column)  
return
```

event RemoveItem (Item as HITEM)

Occurs before removing an item.

| Type | Description |
|---------------|--|
| Item as HITEM | A long expression that specifies the handle of the item being removed. |

The RemoveItem event notifies your application that an item is removed. Use the RemoveItem event to release any extra data associated to the item. Use the [RemoveItem](#) method to remove programmatically an item. The [RemoveColumn](#) event is fired when a column is removed from the control's columns collection. Use the [RemoveAllItems](#) method to remove all items in the control. 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 Item)
{
}
```

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

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

```
C# private void RemoveItem(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_RemoveItemEvent e)
{
}
```

```
C++ void OnRemoveItem(long Item)
{
}
```

```
C++ Builder void __fastcall RemoveItem(TObject *Sender,Excomboboxlib_tlb::HITEM Item)
{
}
```

Delphi

```
procedure Removeltem(ASender: TObject; Item : HITEM);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure Removeltem(sender: System.Object; e:  
AxEXCOMBOBOXLib._IComboBoxEvents_RemoveltemEvent);  
begin  
end;
```

Power...

```
begin event Removeltem(long Item)  
end event Removeltem
```

VB.NET

```
Private Sub Removeltem(ByVal sender As System.Object, ByVal e As  
AxEXCOMBOBOXLib._IComboBoxEvents_RemoveltemEvent) Handles Removeltem  
End Sub
```

VB6

```
Private Sub Removeltem(ByVal Item As EXCOMBOBOXLibCtl.HITEM)  
End Sub
```

VBA

```
Private Sub Removeltem(ByVal Item As Long)  
End Sub
```

VFP

```
LPARAMETERS Item
```

Xbas...

```
PROCEDURE OnRemoveltem(oComboBox,Item)  
RETURN
```

Syntax for Removeltem event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="Removeltem(Item)" LANGUAGE="JScript">  
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">  
Function Removeltem(Item)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComRemoveItem HITEM Item  
    Forward Send OnComRemoveItem Item  
End_Procedure
```

Visual
Objects

```
METHOD OCX_RemoveItem(Item) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_RemoveItem(int _Item)  
{  
}
```

XBasic

```
function RemoveItem as v (Item as OLE::Exontrol.ComboBox.1::HITEM)  
end function
```

dBASE

```
function nativeObject_RemoveItem(Item)  
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 [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 [ScrollPartCaption](#) 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 [Background](#) property to change the visual appearance for any part in the control's scroll bar.

The following VB sample displays the identifier of the scroll's button being clicked:

```
With ComboBox1
    .BeginUpdate
        .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
        .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img> </img> 1"
        .ScrollPartCaption(exVScroll, exRightB1Part) = "<img> </img> 2"
    .EndUpdate
End With
```

```
Private Sub ComboBox1_ScrollButtonClick(ByVal ScrollPart As
EXCOMBOBOXLibCtl.ScrollPartEnum)
    MsgBox (ScrollPart)
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxComboBox1
    .BeginUpdate()
    .set_ScrollPartVisible(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,
```

```

EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part Or
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, True)
    .set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part, " <img> </img> 1")
    .set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, " <img> </img> 2")
    .EndUpdate()
End With

```

```

Private Sub AxComboBox1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As
AxEXCOMBOBOXLib._IComboBoxEvents_ScrollButtonClickEvent) Handles
AxComboBox1.ScrollButtonClick
    MessageBox.Show( e.scrollPart.ToString())
End Sub

```

The following C# sample displays the identifier of the scroll's button being clicked:

```

axComboBox1.BeginUpdate();
axComboBox1.set_ScrollPartVisible(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part |
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, true);
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,
EXCOMBOBOXLib.ScrollPartEnum.exLeftB1Part , " <img> </img> 1");
axComboBox1.set_ScrollPartCaption(EXCOMBOBOXLib.ScrollBarEnum.exVScroll,
EXCOMBOBOXLib.ScrollPartEnum.exRightB1Part, " <img> </img> 2");
axComboBox1.EndUpdate();

```

```

private void axComboBox1_ScrollButtonClick(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_ScrollButtonClickEvent e)
{
    MessageBox.Show(e.scrollPart.ToString());
}

```

The following C++ sample displays the identifier of the scroll's button being clicked:

```

m_comboBox.BeginUpdate();
m_comboBox.SetScrollPartVisible( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );

```

```
m_comboBox.SetScrollPartCaption( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ , _T("<img>  
</img>1"));  
m_comboBox.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img>  
</img>2"));  
m_comboBox.EndUpdate();
```

```
void OnScrollButtonClickComboBox1(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.ComboBox1  
    .BeginUpdate  
        .ScrollPartVisible(0, bitor(32768,32)) = .t.  
        .ScrollPartCaption(0,32768) = "<img> </img> 1"  
        .ScrollPartCaption(0, 32) = "<img> </img> 2"  
    .EndUpdate  
EndWith
```

*** ActiveX Control Event ***

LPARAMETERS scrollpart

```
wait window nowait ltrim(str(scrollpart))
```

Syntax for ScrollButtonClick event, **NET** version, on:

```
C# private void ScrollButtonClick(object  
sender,exontrol.EXCOMBOBOXLib.ScrollBarEnum  
ScrollBar,exontrol.EXCOMBOBOXLib.ScrollPartEnum ScrollPart)  
{  
}
```

```
VB Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As  
exontrol.EXCOMBOBOXLib.ScrollBarEnum,ByVal ScrollPart As  
exontrol.EXCOMBOBOXLib.ScrollPartEnum) Handles ScrollButtonClick
```

End Sub

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

```
C# private void ScrollButtonClick(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_ScrollButtonClickEvent e)
{
}
```

```
C++ void OnScrollButtonClick(long ScrollBar,long ScrollPart)
{
}
```

```
C++ Builder void __fastcall ScrollButtonClick(TObject
*Sender,Excomboboxlib_tlb::ScrollBarEnum
ScrollBar,Excomboboxlib_tlb::ScrollPartEnum ScrollPart)
{
}
```

```
Delphi procedure ScrollButtonClick(ASender: TObject; ScrollBar :
ScrollBarEnum;ScrollPart : ScrollPartEnum);
begin
end;
```

```
Delphi 8 (.NET only) procedure ScrollButtonClick(sender: System.Object; e:
AxEXCOMBOBOXLib._IComboBoxEvents_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
AxEXCOMBOBOXLib._IComboBoxEvents_ScrollButtonClickEvent) Handles
ScrollButtonClick
End Sub
```

```
VB6 Private Sub ScrollButtonClick(ByVal ScrollBar As
```

```
EXCOMBOBOXLibCtl.ScrollBarEnum,ByVal ScrollPart As  
EXCOMBOBOXLibCtl.ScrollPartEnum)  
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)  
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart
```

Xbas...

```
PROCEDURE OnScrollButtonClick(oComboBox,ScrollBar,ScrollPart)  
RETURN
```

Syntax for ScrollButtonClick event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="ScrollButtonClick(ScrollBar,ScrollPart)" LANGUAGE="JScript">  
</SCRIPT>
```

VBS...

```
<SCRIPT LANGUAGE="VBScript">  
Function ScrollButtonClick(ScrollBar,ScrollPart)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComScrollButtonClick OLEScrollBarEnum IIScrollBar  
OLEScrollPartEnum IIScrollPart  
    Forward Send OnComScrollButtonClick IIScrollBar IIScrollPart  
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.ComboBox.1::ScrollBarEnum,ScrollPart as  
OLE::Exontrol.ComboBox.1::ScrollPartEnum)
```

```
end function
```

dBASE

```
function nativeObject_ScrollButtonClick(ScrollBar,ScrollPart)
```

```
return
```

event SelectionChanged ()

Fired after a new item has been selected.

Type

Description

The Selection changed event notifies your application that a new item is selected. Use the [SelectedItem](#) property to determine the handle of the selected item. Use the [Select](#) property to determine the selected value on a specific column. Use the [Value](#) property to specify the selected value on a single column control. Use the [SelectItem](#) property to select an item given its handle. The [Change](#) event is fired if the drop-down portion of the control is visible, and the user clicks an item or presses the Enter key. The [SelectableItem](#) property specifies whether the user can select an item. The [SearchColumnIndex](#) property indicates the index of the column that has the focus. The [AssignEditImageOnSelect](#) property specifies whether the control displays the cell's icon when selection is changed. The [AssignEditImageOnSelect](#) property has effect if the [Style](#) property is DropDown. The [Click](#) event is fired, if the user selects a new item, and the [FireClickOnSelect](#) property is True.

The following samples show options to get the selected item when the user changes the selected item:

The following VB sample displays the selected item, using the Select property:

```
Private Sub ComboBox1_SelectionChanged()  
    With ComboBox1  
        Debug.Print .Select(.SearchColumnIndex)  
    End With  
End Sub
```

The following VB sample displays the selected item, using the Items.SelectedItem property:

```
Private Sub ComboBox1_SelectionChanged()  
    With ComboBox1.Items  
        Dim h As HITEM  
        h = .SelectedItem()  
        If (Not h = 0) Then  
            Debug.Print .CellCaption(h, 0)  
        End If  
    End With  
End Sub
```

The following C++ sample displays the selected item, using the Select property:

```
void OnSelectionChangedCombobox1()
{
    OutputDebugString( m_combobox.GetEditText(
COleVariant(m_combobox.GetSearchColumnIndex() ) ) );
}
```

The following C++ sample displays the selected item, using the Items.SelectedItem property:

```
void OnSelectionChangedCombobox1()
{
    CItems items = m_combobox.GetItems();
    long hItem = items.GetSelectedItem( 0 );
    if ( hItem != 0 )
        OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
}
```

where the V2S function converts a VARIANT value to a string,

```
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 the selected item, using the Select property:

```
Private Sub AxComboBox1_SelectionChanged(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles AxComboBox1.SelectionChanged
    With AxComboBox1
        Debug.WriteLine(.get_Select(.SearchColumnIndex))
    End With
End Sub
```

The following VB.NET sample displays the selected item, using the `Items.SelectedItem` property:

```
Private Sub AxComboBox1_SelectionChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxComboBox1.SelectionChanged
    With AxComboBox1.Items
        Dim h As Integer = .SelectedItem()
        If (Not h = 0) Then
            Debug.WriteLine(.CellCaption(h, 0))
        End If
    End With
End Sub
```

The following C# sample displays the selected item, using the `Select` property:

```
private void axComboBox1_SelectionChanged(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axComboBox1.get_Select(axComboBox1.SearchColumnIndex));
}
```

The following C# sample displays the selected item, using the `Items.SelectedItem` property:

```
private void axComboBox1_SelectionChanged(object sender, EventArgs e)
{
    EXCOMBOBOXLib.Items items = axComboBox1.Items;
    int hItem = items.get_SelectedItem(0);
    if ( hItem != 0 )
        System.Diagnostics.Debug.WriteLine(items.get_CellCaption(hItem,0).ToString());
}
```

The following VFP sample displays the selected item, using the `Select` property:

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

```
With thisform.ComboBox1
```

```
    wait window nowait .Object.Select(.SearchColumnIndex())
```

```
EndWith
```

The following VFP sample displays the selected item, using the `Items.SelectedItem` property:

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

```
With thisform.ComboBox1.Items
```

```
    .DefaultItem = .SelectedItem(0)
```

```
    If (.DefaultItem <> 0) Then
```

```
        wait window nowait .CellCaption(0, 0)
```

```
    EndIf
```

```
EndWith
```

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;
```

Power...

```
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(oComboBox)  
RETURN
```

Syntax for SelectionChanged event, **ICOM** version (others), on:

Java...

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

VBS...

```
<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
```

event Sort ()

Fired when the control sorts a column. */*not supported in the lite version*/*

Type

Description

The control fires the Sort event when the control sorts a column (the user clicks the column's header) 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.

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

```
Private Sub ComboBox1_Sort()  
    Dim s As String, i As Long, c As Column  
    i = 0  
    With ComboBox1.Columns  
        Set c = .ItemBySortPosition(i)  
        While (Not c Is Nothing)  
            s = s + "" & c.Caption & "" " & If(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 OnSortComboBox1()  
{  
    CString strOutput;  
    CColumns columns = m_combobox.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 ) );
}
OutputDebugString( strOutput );
}

```

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

```

Private Sub AxComboBox1_Sort(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxComboBox1.Sort
    With AxComboBox1
        Dim s As String, i As Integer, c As EXCOMBOBOXLib.Column
        i = 0
        With AxComboBox1.Columns
            c = .ItemBySortPosition(i)
            While (Not c Is Nothing)
                s = s + "\"\" & c.Caption & \" \" & If(c.SortOrder =
EXCOMBOBOXLib.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 C# sample displays the list of columns being sorted:

```

private void axComboBox1_Sort(object sender, System.EventArgs e)
{
    string strOutput = "";
    int i = 0;

```

```

EXCOMBOBOXLib.Column column = axComboBox1.Columns.get_ItemBySortPosition( i
);
while ( column != null )
{
    strOutput += column.Caption + " " + ( column.SortOrder ==
EXCOMBOBOXLib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
    column = axComboBox1.Columns.get_ItemBySortPosition( ++i );
}
Debug.WriteLine( strOutput );
}

```

The following VFP sample displays the list of columns being sorted (the code is listed in the Sort event) :

```

local s, i, c
i = 0
s = ""
With thisform.ComboBox1.Columns
    c = .ItemBySortPosition(i)
    do While (!isnull(c))
        with c
            s = s + "" + .Caption
            s = s + " " + If(.SortOrder = 1, "A", "D") + " "
            i = i + 1
        endwhile
        c = .ItemBySortPosition(i)
    enddo
endwith
s = "Sort: " + s
wait window nowait s

```

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, **ICOM** 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;
```

```
Power... 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(oComboBox)
RETURN
```

Syntax for Sort event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="Sort()" LANGUAGE="JScript">
</SCRIPT>
```

```
VBS... <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
```

event ToolTip (Item as HITEM, 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 |
|--------------------|---|
| Item as HITEM | A long expression that indicates the item's handle or 0 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 window. The x values is always expressed in screen coordinates. |
| Y as Long | A long expression that indicates the top location of the tooltip window. The y values is always expressed in screen coordinates. |
| CX as Long | long expression that indicates the width of the tooltip window. |
| CY as Long | A long expression that indicates the height of the tooltip 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 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 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 Item,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 Item 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
```

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

```
C# private void ToolTip(object sender,
AxEXCOMBOBOXLib._IComboBoxEvents_ToolTipEvent e)
{
}
```

```
C++ void OnToolTip(long Item,long ColIndex,BOOL FAR* Visible,long FAR* X,long FAR*
Y,long CX,long CY)
{
}
```

```
C++ Builder void __fastcall ToolTip(TObject *Sender,Excomboboxlib_tlb::HITEM Item,long
ColIndex,VARIANT_BOOL * Visible,long * X,long * Y,long CX,long CY)
{
}
```

```
Delphi procedure ToolTip(ASender: TObject; Item : HITEM;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:
AxEXCOMBOBOXLib._IComboBoxEvents_ToolTipEvent);
begin
end;
```

```
Power... begin event ToolTip(long Item,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
AxEXCOMBOBOXLib._IComboBoxEvents_ToolTipEvent) Handles ToolTip
End Sub
```

```
VB6 Private Sub ToolTip(ByVal Item As EXCOMBOBOXLibCtl.HITEM,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 Item 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 Item,ColIndex,Visible,X,Y,CX,CY
```

```
Xbas... PROCEDURE OnToolTip(oComboBox,Item,ColIndex,Visible,X,Y,CX,CY)
RETURN
```

Syntax for ToolTip event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="ToolTip(Item,ColIndex,Visible,X,Y,CX,CY)"
LANGUAGE="JScript">
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">
Function ToolTip(Item,ColIndex,Visible,X,Y,CX,CY)
End Function
</SCRIPT>
```

```
Visual Data... Procedure OnComToolTip HITEM IItem Integer IColIndex Boolean IVisible Integer
IIX Integer IY Integer ICX Integer ICY
Forward Send OnComToolTip IItem IColIndex IVisible IIX IY ICX ICY
End_Procedure
```

```
Visual Objects METHOD OCX_ToolTip(Item,ColIndex,Visible,X,Y,CX,CY) CLASS MainDialog
RETURN NIL
```

```
X++ void onEvent_ToolTip(int _Item,int _ColIndex,COMVariant /*bool*/
_Visible,COMVariant /*long*/ _X,COMVariant /*long*/ _Y,int _CX,int _CY)
{
}
```

```
XBasic function ToolTip as v (Item as OLE::Exontrol.ComboBox.1::HITEM,ColIndex as
```

```
N,Visible as L,X as N,Y as N,CX as N,CY as N)
```

```
end function
```

dBASE

```
function nativeObject_ToolTip(Item,ColIndex,Visible,X,Y,CX,CY)
```

```
return
```

OwnerDrawHandler object

The OwnerDrawHandler interface provides an elegant way to let user paints the cell. The [CellOwnerDraw](#) property requires an object that implements the OwnerDrawHandler interface. Use the [Def\(exCellOwneDraw\)](#) property to assign an owner draw object for the entire column. The control calls DrawCell method when an owner draw cell requires painting. The interface definition is like follows:

```
[
  uuid(29301C67-ABB7-4E36-9C4D-C4DB8479D893),
  pointer_default(unique)
]
interface IOwnerDrawHandler : IUnknown
{
  [id(1), helpstring("The source paints the cell.")] HRESULT DrawCell( long hDC, long
left, long top, long right, long bottom, long Item, long Column, IDispatch* Source );
}
```

The following VB sample shows how to paint a gradient color into the cells:

```
Option Explicit
Implements EXCOMBOBOXLibCtl.IOwnerDrawHandler ' The form implements the
IOwnerDrawHandler interface

Private Type RECT
  left As Long
  top As Long
  right As Long
  bottom As Long
End Type

Private Const ETO_OPAQUE = 2
Private Declare Sub InflateRect Lib "user32" (lpRect As RECT, ByVal x As Long, ByVal Y As
Long)
Private Declare Function ExtTextOut Lib "gdi32" Alias "ExtTextOutA" (ByVal hDC As Long,
ByVal x As Long, ByVal Y As Long, ByVal wOptions As Long, lpRect As RECT, ByVal lpString
As String, ByVal nCount As Long, lpDx As Long) As Long
Private Declare Function SetBkColor Lib "gdi32" (ByVal hDC As Long, ByVal crColor As
Long) As Long
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hDC As Long,
```

```
ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As Long
```

```
Private Declare Function SetTextColor Lib "gdi32" (ByVal hDC As Long, ByVal crColor As Long) As Long
```

```
Private Declare Function OleTranslateColor Lib "olepro32" (ByVal c As Long, ByVal p As Long, c As Long) As Long
```

```
Private Const DT_VCENTER = &H4
```

```
Private Const DT_CENTER = &H1
```

```
Private Const DT_NOPREFIX = &H800
```

```
Private Sub DrawGradient(ByVal hDC As Long, ByVal left As Long, ByVal top As Long, ByVal right As Long, ByVal bottom As Long, ByVal c1 As Long, ByVal c2 As Long)
```

```
    On Error Resume Next
```

```
    Dim x As Long, rg, gg, bg, r1, r2, g1, g2, b1, b2
```

```
    Dim rc As RECT
```

```
    With rc
```

```
        .left = left
```

```
        .right = right
```

```
        .top = top
```

```
        .bottom = bottom
```

```
    End With
```

```
    OleTranslateColor c1, 0, c1
```

```
    OleTranslateColor c2, 0, c2
```

```
    r1 = c1 Mod 256
```

```
    r2 = c2 Mod 256
```

```
    b1 = Int(c1 / 65536)
```

```
    b2 = Int(c2 / 65536)
```

```
    g1 = Int(c1 / 256) Mod 256
```

```
    g2 = Int(c2 / 256) Mod 256
```

```
    For x = left To right Step 2
```

```
        rc.left = x
```

```
        SetBkColor hDC, RGB(r1 + (x - left) * (r2 - r1) / (right - left), g1 + (x - left) * (g2 - g1) / (right - left), b1 + (x - left) * (b2 - b1) / (right - left))
```

```
        ExtTextOut hDC, rc.left, rc.top, ETO_OPAQUE, rc, " ", 1, x
```

```
    Next
```

```
End Sub
```

```
Private Sub ComboBox1_InsertItem(Index As Integer, ByVal Item As  
EXCOMBOBOXLibCtl.HITEM)
```

```
    With ComboBox1(Index).Items
```

```
        Set .CellOwnerDraw(Item, 2) = Me      ' The form implements the
```

```
IOwnerDrawHandler interface
```

```
    End With
```

```
End Sub
```

```
Private Sub IOwnerDrawHandler_DrawCell(ByVal hDC As Long, ByVal left As Long, ByVal  
top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As Long, ByVal  
Column As Long, ByVal Source As Object)
```

```
    With Source.Items
```

```
        DrawGradient hDC, left, top, right / 2, bottom, .CellCaption(Item, 0),
```

```
.CellCaption(Item, 1)
```

```
        DrawGradient hDC, right / 2, top, right, bottom, .CellCaption(Item, 1),
```

```
.CellCaption(Item, 0)
```

```
        Dim rc As RECT
```

```
        With rc
```

```
            .left = left + 16
```

```
            .right = right - 1
```

```
            .top = top + 1
```

```
            .bottom = bottom - 1
```

```
        End With
```

```
        Dim c As Long, r, g, b
```

```
        c = .CellCaption(Item, 1)
```

```
        r = c Mod 256
```

```
        b = Int(c / 65536)
```

```
        g = Int(c / 256) Mod 256
```

```
        Dim str As String
```

```
        str = If(r > 0, r, If(g > 0, g, If(b > 0, b, 0)))
```

```
        SetTextColor hDC, vbWhite
```

```
        DrawText hDC, str, Len(str), rc, DT_VCENTER Or DT_NOPREFIX Or DT_CENTER
```

```
    End With
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    Dim j As Long
```

```

For j = 0 To 2
  With ComboBox1(j)
    .BeginUpdate
    .ColumnAutoSize = True
    With .Columns
      With .Add("From")
        .Visible = False
      End With
      With .Add("To")
        .Visible = False
      End With
      With .Add("Gradient")
        .Width = 128
      End With
    End With
  End With

  With .Items
    Dim i As Long, h As HITEM
    For i = 255 To 0 Step -1
      h = .AddItem(vbWhite)
      .CellCaption(h, 1) = RGB(i * If(j = 0, 1, 0), i * If(j = 1, 1, 0), i * If(j = 2, 1, 0))
    Next
    .SelectItem(.ItemByIndex(0)) = True
  End With

  .EndUpdate
End With
Next
End Sub

```

Name

Description

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** (remainder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

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

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

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

- 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 not found, else the associated element in the collection if it is found. The syntax for array operator is

expression array (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *month(value)-1 array ('J','F','M','A','M','Jun','J','A','S','O','N','D')* is equivalent with *month(value)-1 case (default:"; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D')*.

- **in (include operator)**, specifies whether an element is found in a set of constant elements. The in operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for in operator is

expression in (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *value in (11,22,33,44,13)* is equivalent with *(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)*. The in operator is not a time consuming as the equivalent or version is, so when you have large number of constant elements it is recommended using the in operator. Shortly, if the collection of elements has 1000 elements the in operator could take up to 8 operations in order to find if an element fits the set, else if the or

statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- **switch** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the *c1*, *c2*, ... are constant elements, and the *default* is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "*%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))*". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than *iif* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of *n* expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary *case()* operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (*c1*, *c2*, ...). For instance, if the value of expression is not any of *c1*, *c2*, the *default_expression* is executed and returned. If the value of the expression is *c1*, then the *case()* operator executes and returns the *expression1*. The *default*, *c1*, *c2*, *c3*, ... must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1 ; #4/1/2002#:1 ; #5/1/2002#:1)* indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

- #4/1/2009#, from hours 06:00 AM to 12:00 PM
- #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
- #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using *if* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. The type operator may return any of the following: 0 - empty (not initialized), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, 7 - **date**, 8 - **string**, 9 - object, 10 - error, 11 - **boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bits. For instance `type(%1) = 8` specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the `str(-12.54)` returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the `dbl("12.54")` returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the `date(`)` gets the current date (no time included), the `date(now)` gets the current date-time, while the `date("01/01/2001")` returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the `dateS("01/01/2001 14:00:00")` returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, `hex(`FF`)` returns 255, while the `hex(255)` or `hex(0xFF)` returns the `FF` string. The `hex(hex(`FFFFFFFF`))` always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

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

2)

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

The operators for numbers are:

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