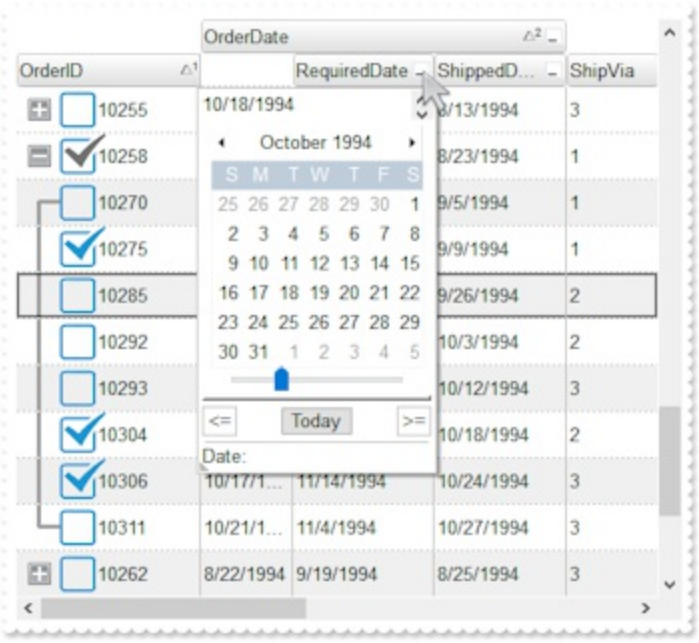
 ExTree

Add an enhanced ExTree component to your application. The ExTree provides the entire range of capabilities you would expect to see in a state-of-the-art Tree component. The ExTree control simulates a simple tree, a multi-column tree, a list or a listview control. The exTree component lets the user changes its visual appearance using **skins**, each one providing an additional visual experience that enhances viewing pleasure. Skins are relatively easy to build and put on any part of the control.

Features of the full version include:

- **Print** and Print Preview support.
- **WYSWYG Template/Layout Editor** support.
- **Skinnable Interface** support (ability to apply a skin to the any background part, including the scroll bars)
- **ActiveX hosting** (you can place any ActiveX component in any item of the tree)
- **Multiple Levels Header** support
- **Sorting by Single or Multiple Columns** support
- 'starts with' and 'contains' **incremental searching** support
- ADO and DAO support
- **Filter** support
  - **Filter-Prompt** support, allows you to filter the items as you type while the filter bar is always visible on the bottom part of the list area.
  - **Filter-On-Type** support**.** Ability to filter items by a column, as you type.
  - Ability to filter items using patterns that include **wild card characters** like *, ? or #, **items between two dates, numbers**, **checkboxes** with an easy UI interface.
  - Ability to filter items using **OR, AND** or **NOT operators** between columns.
- **Conditional Format** support
- **Computed Fields** support
- Ability to load and generate template files.
- OLE drag and drop support
- Multi-lines tooltip support, XP shadow effect
- Mouse wheel support
- Multiple selection
- User resizable columns
- Formatting columns support
- Drag and Drop columns support.
- Events from contained components are fired through to your program using the exact same model used in VB
- **Locked/Fixed columns** support
- Ability to load icons or pictures from BASE64 encoded strings

- **Multi-lines items** support
- Ability to assign multiple icons to a cell.
- Any cell supports **Built-in HTML format.**
- Partial Check Support. Built-in checkbox reflection to reflect that state of children, parents.
- **Locked/Fixed items** support.
- **"Split Cells"** support.
- **"Merge Cells"** support.
- **Divider items** support
- **Not selectable** items support
- Unlimited color options for cells, items
- Different font for any cell or item
- Cells can be formatted individually, or via columns or rows
- Radio buttons, images, check boxes
- Column alignment right, left, or center
- Hot Tracking Support.
- Background Picture Support.
- Transparent Selection Support.
- and more



How to get support?

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

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

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

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

Regards,
Exontrol Development Team

https://www.exontrol.com

How to start?

The following steps show you progressively how to start programming the Exontrol's ExTree 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 or Items.InsertControlItem to insert child items, or child items that host ActiveX control, when you need to display your data as a tree. The PutItems method takes an array of data and fetches in the control. The AddItem 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.

constants AlignmentEnum

The Column object uses the AlignmentEnum enumeration to align a column. See the Alignment property of the Column for more details.

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

constants AppearanceEnum

The AppearanceEnum enumeration is used to specify the appearance of the control's header bar. See also the HeaderAppearance property.

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

constants AutoDragEnum

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

The AutoDragEnum type supports the following values:

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

exAutoDragPositionKeepIndent	2	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.
exAutoDragPositionOnShortTouch	256	exAutoDragPositionOnShortTouch. The object can be dragged from a position to another, but not

		outside of its group.
exAutoDragPositionKeepIndentOnShortTouch	512	exAutoDragPositionKeepIndentOnShortTouch. The object can be dragged to any position or to any parent, while the dragging object keeps its indentation.
exAutoDragPositionAnyOnShortTouch	768	exAutoDragPositionAnyOnShortTouch. The object can be dragged to any position or to any parent, with no restriction.
exAutoDragCopyOnShortTouch	2048	exAutoDragCopyOnShortTouch. Drag and drop the selected objects to a target application, and paste them as image or text.
exAutoDragCopyTextOnShortTouch	2304	exAutoDragCopyTextOnShortTouch. Drag and drop the selected objects to a target application, and paste them as text only.
exAutoDragCopyImageOnShortTouch	2560	exAutoDragCopyImageOnShortTouch. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnShortTouch	2816	exAutoDragCopySnapShotOnShortTouch. Drag and drop a snap shot of the current component.
exAutoDragScrollOnShortTouch	4096	exAutoDragScrollOnShortTouch. The component is scrolled by clicking the object and dragging to a new position.
exAutoDragPositionOnRight	65536	exAutoDragPositionOnRight. The object can be dragged from a position to another, but not outside of its group.
exAutoDragPositionKeepIndentOnRight	131072	exAutoDragPositionKeepIndentOnRight. The object can be dragged to any position or to any parent, while the dragging object keeps its indentation.
exAutoDragPositionAnyOnRight	196608	exAutoDragPositionAnyOnRight. The object can be dragged to any position or to any parent, with no restriction.
exAutoDragCopyOnRight	524288	exAutoDragCopyOnRight. Drag and drop the selected objects to a target application, and paste them as image or text.
exAutoDragCopyTextOnRight	589824	exAutoDragCopyTextOnRight. Drag and drop the selected objects to a target application, and paste them as text only.
		exAutoDragCopyImageOnRight. Drag and drop the

exAutoDragCopyImageOnRight	655360	selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnRight	720896	exAutoDragCopySnapShotOnRight. Drag and drop a snap shot of the current component.
exAutoDragScrollOnRight	1048576	exAutoDragScrollOnRight. The component is scrolled by clicking the object and dragging to a new position.
exAutoDragPositionOnLongTouch	16777216	exAutoDragPositionOnLongTouch. The object can be dragged from a position to another, but not outside of its group.
exAutoDragPositionKeepIndentOnLongTouch	33554432	exAutoDragPositionKeepIndentOnLongTouch. The object can be dragged to any position or to any parent, while the dragging object keeps its indentation.
exAutoDragPositionAnyOnLongTouch	67108864	exAutoDragPositionAnyOnLongTouch. The object can be dragged to any position or to any parent, with no restriction.
exAutoDragCopyOnLongTouch	134217728	exAutoDragCopyOnLongTouch. Drag and drop the selected objects to a target application, and paste them as image or text.
exAutoDragCopyTextOnLongTouch	150994944	exAutoDragCopyTextOnLongTouch. Drag and drop the selected objects to a target application, and paste them as text only.
exAutoDragCopyImageOnLongTouch	167772160	exAutoDragCopyImageOnLongTouch. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnLongTouch	184549376	exAutoDragCopySnapShotOnLongTouch. Drag and drop a snap shot of the current component.
exAutoDragScrollOnLongTouch	268435456	exAutoDragScrollOnLongTouch. The component is scrolled by clicking the object and dragging to a new position.

constants AutoSearchEnum

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

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

constants BackgroundPartEnum

/*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, when it is up. 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.

exDateHeader	8	Specifies the visual appearance for the header in a calendar control.
exDateTodayUp	9	Specifies the visual appearance for the today button in a calendar control, when it is up.
exDateTodayDown	10	Specifies the visual appearance for the today button in a calendar control, when it is down.
exDateScrollThumb	11	Specifies the visual appearance for the scrolling thumb in a calendar control.
exDateScrollRange	12	Specifies the visual appearance for the scrolling range in a calendar control.
exDateSeparatorBar	13	Specifies the visual appearance for the separator bar in a calendar control.
exDateSelect	14	Specifies the visual appearance for the selected date in a calendar control.
exSelBackColorFilter	20	Specifies the visual appearance for the selection in the drop down filter window. The drop down filter window shows up when the user clicks the filter button in the column's header. Use the DisplayFilterButton property to specify whether the drop down filter bar button is visible or hidden.
exSelForeColorFilter	21	Specifies the foreground color for the selection in the drop down filter window.
exBackColorFilter	26	Specifies the background color for the drop down filter window. If not specified, the BackColorHeader property specifies the drop down filter's background color. Use the exSelBackColorFilter option to specify the selection background visual appearance in the drop down filter window. If not specified, the ForeColorHeader property specifies the drop down filter's foreground color. Use the exSelForeColorFilter option to specify the selection foreground color in the drop down filter window.
exForeColorFilter	27	Specifies the foreground color for the drop down filter window.
exSortBarLinkColor	28	Indicates the color or the visual appearance of the links between columns in the control's sort bar.
		Specifies the visual appearance for the column when the cursor hovers the column. By default, the

exCursorHoverColumn	32	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.
exDragDropBefore	33	Specifies the visual appearance for the drag and drop cursor before showing the items. This option can be used to apply a background to the dragging items, before painting the items.
exDragDropAfter	34	Specifies the visual appearance for the drag and drop cursor after showing the items. This option can be used to apply a semi-transparent/opaque background to the dragging items, after painting the items. If the exDragDropAfter option is set on white (0x00FFFFFF), the image is not showing on OLE Drag and drop.
exDragDropListTop	35	Specifies the graphic feedback of the item from the drag and drop cursor if the cursor is in the top half of the row. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
exDragDropListBottom	36	Specifies the graphic feedback of the item from the drag and drop cursor if the cursor is in the bottom half of the row. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint*

		property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.
exDragDropForeColor	37	Specifies the foreground color for the items being dragged. By default, the foreground color is black.
exDragDropListOver	38	Specifies the graphic feedback of the item from the cursor if it is over the item. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
exDragDropListBetween	39	Specifies the graphic feedback of the item when the drag and drop cursor is between items. *Please note, that if a visual effect is specified for exDragDropListOver AND exDragDropListBetween states, and a visual effect is specified for exDragDropListTop OR/AND exDragDropListBottom state(s), the exDragDropListTop visual effect is displayed ONLY if the cursor is over the first visible item, and the exDragDropListBottom visual effect is shown ONLY for the last visible item. Use the ItemFromPoint property to retrieve the hit test code for the part from the cursor. This option can be changed during the OLEDragOver event to change the visual effect for the item from the cursor at runtime.*
		Specifies the alignment of the drag and drop image relative to the cursor. By default, the exDragDropAlign option is 0, which initially the drag and drop image is shown centered relative to the position of the cursor.

exDragDropAlign	40	The valid values are listed as follows (hexa representation): • 0x00000000, (default), the drag and drop image is shown centered relative to the cursor, and shows up. • 0x01000000, (left), the drag and drop image is shown to the left of the cursor. • 0x02000000, (right), the drag and drop image is shown to the right of the cursor. • 0x04000000, (center-down), the drag and drop image is shown centered relative to the cursor, and shows down. • 0xFF000000, (as- is), the drag and drop image is shown as it is clicked.
exHeaderFilterBarActive	41	exHeaderFilterBarActive. Specifies the visual appearance of the drop down filter bar button, while filter is applied to the column.
exToolTipAppearance	64	Indicates the visual appearance of the borders of the tooltips. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the CellToolTip property to specify the cell's tooltip. Use the ToolTipWidth property to specify the width of the tooltip window. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ShowToolTip method to display a custom tooltip.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.
exColumnsFloatBackColor	87	Specifies the background color for the Columns float bar.
exColumnsFloatScrollBackColor	88	Specifies the background color for the scroll bars in the Columns float bar.
exColumnsFloatScrollPressBackColor	89	Specifies the background color for the scroll bars in the Columns float bar, while the scroll part is pressed.
exColumnsFloatScrollUp	90	Specifies the visual appearance of the up scroll bar.

exColumnsFloatScrollDown	91	Specifies the visual appearance of the down scroll bar.
exColumnsFloatAppearance	92	Specifies the visual appearance for the frame/borders of the Column's float bar
exColumnsFloatCaptionBackColor	93	Specifies the visual appearance for caption, if the Background(exColumnsFloatAppearance) property is specified.
exColumnsFloatCaptionForeColor	94	Specifies the foreground color for the caption, if the Background(exColumnsFloatAppearance) property is specified.
exColumnsFloatCloseButton	95	Specifies the visual appearance for the closing button, if the Background(exColumnsFloatAppearance) property is specified.
exListOLEDropPosition	96	By default, the exListOLEDropPosition is 0, which means no effect. Specifies the visual appearance of the dropping position inside the control, when the control is implied in a OLE Drag and Drop operation. The exListOLEDropPosition has effect only if different than 0, and the OLEDropMode property is not exOLEDropNone. For instance, set the Background(exScheduleOLEDropPosition) property on RGB(0,0,255), and a blue line is shown at the item position when the cursor is hover the control, during an OLE Drag and Drop position. The OLEDragDrop event notifies your application once an object is drop in the control.
exSelBackColorHide	166	exSelBackColorHide. Specifies the selection's background color, when the control has no focus.
exSelForeColorHide	167	exSelForeColorHide. Specifies the selection's foreground color, when the control has no focus.
exTreeGlyphOpen	180	exTreeGlyphOpen. Specifies the visual appearance for the +/- buttons when it is collapsed.
exTreeGlyphClose	181	exTreeGlyphClose. Specifies the visual appearance for the +/- buttons when it is expanded.
exColumnsPositionSign	182	exColumnsPositionSign. Specifies the visual appearance for the position sign between columns, when the user changes the position of the column by drag an drop.

exTreeLinesColor	186	exTreeLinesColor. Specifies the color to show the tree-lines (connecting lines from the parent to the children)
exVSUp	256	The up button in normal state, in the vertical scroll bar.
exVSUpP	257	The up button when it is pressed, in the vertical scroll bar.
exVSUpD	258	The up button when it is disabled, in the vertical scroll bar.
exVSUpH	259	The up button when the cursor hovers it, in the vertical scroll bar.
exVSThumb	260	The thumb part (exThumbPart) in normal state, in the vertical scroll bar.
exVSThumbP	261	The thumb part (exThumbPart) when it is pressed, in the vertical scroll bar.
exVSThumbD	262	The thumb part (exThumbPart) when it is disabled, in the vertical scroll bar.
exVSThumbH	263	The thumb part (exThumbPart) when cursor hovers it, in the vertical scroll bar.
exVSDown	264	The down button in normal state, in the vertical scroll bar.
exVSDownP	265	The down button when it is pressed, in the vertical scroll bar.
exVSDownD	266	The down button when it is disabled, in the vertical scroll bar.
exVSDownH	267	The down button when the cursor hovers it, in the vertical scroll bar.
exVSLower	268	The lower part (exLowerBackPart) in normal state, in the vertical scroll bar.
exVSLowerP	269	The lower part (exLowerBackPart) when it is pressed, in the vertical scroll bar.
exVSLowerD	270	The lower part (exLowerBackPart) when it is disabled, in the vertical scroll bar.
exVSLowerH	271	The lower part (exLowerBackPart) when the cursor hovers it, in the vertical scroll bar.
		The upper part (exUpperBackPart) in normal

exVSUpper	272	state, in the vertical scroll bar.
exVSUpperP	273	The upper part (exUpperBackPart) when it is pressed, in the vertical scroll bar.
exVSUpperD	274	The upper part (exUpperBackPart) when it is disabled, in the vertical scroll bar.
exVSUpperH	275	The upper part (exUpperBackPart) when the cursor hovers it, in the vertical scroll bar.
exVSBack	276	The background part (exLowerBackPart and exUpperBackPart) in normal state, in the vertical scroll bar.
exVSBackP	277	The background part (exLowerBackPart and exUpperBackPart) when it is pressed, in the vertical scroll bar.
exVSBackD	278	The background part (exLowerBackPart and exUpperBackPart) when it is disabled, in the vertical scroll bar.
exVSBackH	279	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it, in the vertical scroll bar.
exHSLeft	384	The left button in normal state, in the horizontal scroll bar.
exHSLeftP	385	The left button when it is pressed, in the horizontal scroll bar..
exHSLeftD	386	The left button when it is disabled, in the horizontal scroll bar..
exHSLeftH	387	The left button when the cursor hovers it, in the horizontal scroll bar..
exHSThumb	388	The thumb part (exThumbPart) in normal state, in the horizontal scroll bar..
exHSThumbP	389	The thumb part (exThumbPart) when it is pressed, in the horizontal scroll bar..
exHSThumbD	390	The thumb part (exThumbPart) when it is disabled, in the horizontal scroll bar..
exHSThumbH	391	The thumb part (exThumbPart) when the cursor hovers it, in the horizontal scroll bar..
exHSRight	392	The right button in normal state, in the horizontal scroll bar..

exHSRightP	393	The right button when it is pressed, in the horizontal scroll bar..
exHSRightD	394	The right button when it is disabled, in the horizontal scroll bar..
exHSRightH	395	The right button when the cursor hovers it, in the horizontal scroll bar..
exHSLower	396	The lower part (exLowerBackPart) in normal state, in the horizontal scroll bar..
exHSLowerP	397	The lower part (exLowerBackPart) when it is pressed, in the horizontal scroll bar..
exHSLowerD	398	The lower part (exLowerBackPart) when it is disabled, in the horizontal scroll bar..
exHSLowerH	399	The lower part (exLowerBackPart) when the cursor hovers it, in the horizontal scroll bar..
exHSUpper	400	The upper part (exUpperBackPart) in normal state, in the horizontal scroll bar..
exHSUpperP	401	The upper part (exUpperBackPart) when it is pressed, in the horizontal scroll bar..
exHSUpperD	402	The upper part (exUpperBackPart) when it is disabled, in the horizontal scroll bar..
exHSUpperH	403	The upper part (exUpperBackPart) when the cursor hovers it, in the horizontal scroll bar..
exHSBack	404	The background part (exLowerBackPart and exUpperBackPart) in normal state, in the horizontal scroll bar..
exHSBackP	405	The background part (exLowerBackPart and exUpperBackPart) when it is pressed, in the horizontal scroll bar..
exHSBackD	406	The background part (exLowerBackPart and exUpperBackPart) when it is disabled, in the horizontal scroll bar..
exHSBackH	407	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it, in the horizontal scroll bar..
exSBtn	324	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), in normal state, in the scroll bar..
		All button parts (L1-L5, LButton, exThumbPart,

exSBtnP	325	RButton, R1-R6), when it is pressed, in the scroll bar..
exSBtnD	326	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is disabled, in the scroll bar..
exSBtnH	327	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when the cursor hovers it , in the scroll bar..
exScrollHoverAll	500	Enables or disables the hover-all feature. By default (Background(exScrollHoverAll) = 0), the left/top, right/bottom and thumb parts of the control' scrollbars are displayed in hover state while the cursor hovers any part of the scroll bar (hover-all feature). The hover-all feature is available on Windows 11 or greater, if only left/top, right/bottom, thumb, lower and upper-background parts of the scrollbar are visible, no custom visual-appearance is applied to any visible part. The hover-all feature is always on If Background(exScrollHoverAll) = -1. The Background(exScrollHoverAll) = 1 disables the hover-all feature.
exVSThumbExt	503	The thumb-extension part in normal state.
exVSThumbExtP	504	The thumb-extension part when it is pressed.
exVSThumbExtD	505	The thumb-extension part when it is disabled.
exVSThumbExtH	506	The thumb-extension when the cursor hovers it.
exHSThumbExt	507	The thumb-extension in normal state.
exHSThumbExtP	508	The thumb-extension when it is pressed.
exHSThumbExtD	509	The thumb-extension when it is disabled.
exHSThumbExtH	510	The thumb-extension when the cursor hovers it.
exScrollSizeGrip	511	The background of the part being displayed in the bottom-right side of the control, if vertical and horizontal scroll bars are visible.

constants BackModeEnum

Specifies how the control displays the selection.

Name	Value	Description
exOpaque	0	The selection is opaque.
exTransparent	1	The selection is transparent.
exGrid	2	The control paints a grid selection.

constants CaptionFormatEnum

The CaptionFormatEnum type defines how the cell's caption is painted. Use the CellCaptionFormat property to specify whether the cell displays built-in HTML format. Use the HTMLCaption property to specify an HTML caption for a column. The CellToolTip property supports also built-in HTML format.

Name	Value	Description
exText	0	No HTML tags are painted.
		The control uses built-in HTML tags to display the caption using HTML format. The control supports the following HTML tags: • ** ... ** displays the text in **bold** • ** ... ** displays the text in *italics* • **<u> ... </u>** underlines the text • **<s> ... </s>** ~~Strike~~-through text • **<a id;options> ... ** displays an anchor element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link.The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements. The control supports expandable HTML captions feature which allows you to expand(show)/collapse(hide) different information using <a ;exp=> or <a ;e64=> anchor tags. The exp/e64 field of the anchor stores the HTML line/lines to show once the user clicks/collapses/expands the caption. ◦ exp, stores the plain text to be shown once the user clicks the anchor, such as " <a ;exp=show lines>" ◦ e64, encodes in BASE64 the HTML text to be shown once the user clicks the anchor,

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

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

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a

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

stretch the picture, else the original size of the picture is used.

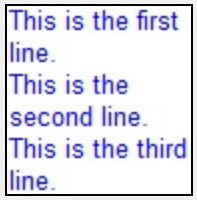
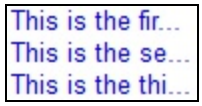
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance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

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb

		represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<**out** 000000><fgcolor=FFFFFF>outlined</fgcolor></**out**>" generates the following picture:  • **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<**sha**>shadow</**sha**>" generates the following picture:  or "*<**sha** 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></**sha**>*" gets: 
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	Indicates that the cell's caption is displayed on a single line. In this case any \r\n or HTML tags is ignored. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as: 
exCaptionWordWrap	0	Specifies that the cell's caption is displayed on multiple lines, by wrapping the words. Any \r\n or HTML tag breaks the line. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as: 
exCaptionBreakWrap	1	Specifies that the cell's caption is displayed on multiple lines, by wrapping the breaks only. Only The \r\n or HTML tag breaks the line. For instance the "This is the first line.\r\nThis is the second line.\r\nThis is the third line." shows as: 

constants CheckStateEnum

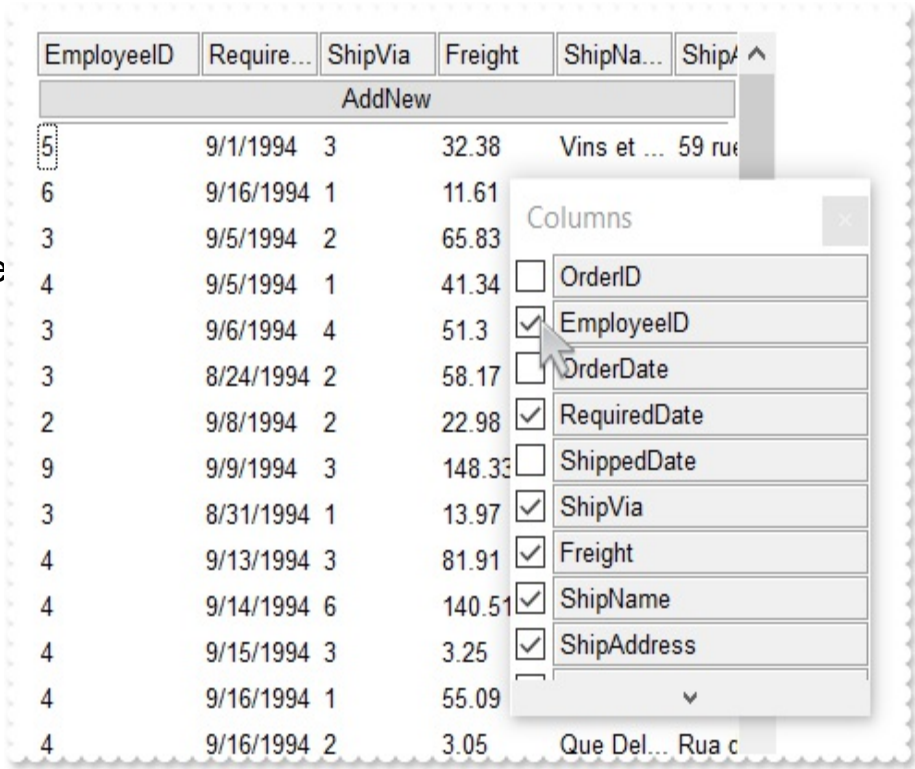
Specifies the states for a checkbox in the control.

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 ColumnsFloatBarVisibleEnum

The ColumnsFloatBarVisibleEnum type specifies whether the control's Columns float-bar is visible or hidden. The ColumnsFloatBarVisibleEnum type supports the following values:

Name	Value	Description
exColumnsFloatBarHidden	0	Indicates that the control's Columns float-panel is not visible (hidden)
exColumnsFloatBarVisibleIncludeHiddenColumns	1	Specifies that the control's Columns float-panel shows only hidden-columns (dragable-columns only). The Visible property specifies whether the column is visible or hidden.
exColumnsFloatBarVisibleIncludeChe	2	Indicates that the control's Columns float-panel shows visible and hidden columns with a check-box associated (dragable-columns only), The Visible property specifies whether the column is visible or hidden.



constants DefColumnEnum

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

Name	Value	Description
exCellHasCheckBox	0	Assigns check boxes to all cells in the column, if it is True. Similar with the CellHasCheckBox property. *(Boolean expression, False)*
exCellHasRadioButton	1	Assigns radio buttons to all cells in the column, if it is True. Similar with the CellHasRadioButton property. *(Boolean expression, False)*
exCellHasButton	2	Specifies that all cells in the column are buttons, if it is True. Similar with the CellHasButton property. *(Boolean expression, False)*
exCellButtonAutoWidth	3	Specifies that all buttons in the column fit the cell's caption, if it is True. Similar with the CellButtonAutoWidth property. *(Boolean expression, False)*
exCellBackColor	4	Specifies the background color for all cells in the column. Use the CellBackColor property to assign a background color for a specific cell. The property has effect only if the property is different than zero. *(Long expression)*
exCellForeColor	5	Specifies the foreground color for all cells in the column. Use the CellForeColor property to assign a foreground color for a specific cell. The property has effect only if the property is different than zero. *(Long expression)*

exCellVAlignment	6	Specifies the column's vertical alignment. By default, the Def(exCellVAlignment) property is 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 the type of text being displayed in the cells in the column. Similar with the CellCaptionFormat property, *(CaptionFormatEnum expression, exText)*
		Specifies the template for the column's filter when

exFilterPatternTemplate	21	the Filter property or the 'Filter For' field is populated. This property customizes the filter pattern for the column when the FilterType property is set to exPattern. It supports the **<%filter%>** keyword, which replaces the original filter input. For example, setting Def(exFilterPatternTemplate) to "*<%filter%>*" filters for all items containing the specified sequence, while setting it to "Item*<%filter%>" filters for all items starting with 'Item' and ending with the typed characters. If the Column.Def(exFilterPatternTemplate) property is empty, the filter is applied as it is (no effect). *(String expression)*
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)*
		Gets or sets the top padding (space) of the cells within the column.

exCellPaddingTop	50	*(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	Gets or sets a value that indicates whether the control's content is updated while the user is resizing the column. *(Boolean expression, False)*

constants DescriptionTypeEnum

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

Name	Value	Description
exFilterBarAll	0	Defines the caption of (All) in the filter bar window. If the Description(exFilterBarAll) property is empty, the (All) predefined item is not shown in the drop down filter window.
exFilterBarBlanks	1	Defines the caption of (Blanks) in the filter bar window. If the Description(exFilterBarBlanks) property is empty, the (Blanks) predefined item is not shown in the drop down filter window.
exFilterBarNonBlanks	2	Defines the caption of (NonBlanks) in the filter bar window. If the Description(exFilterBarNonBlanks) property is empty, the (NonBlanks) predefined item is not shown in the drop down filter window.
exFilterBarFilterForCaption	3	Defines the caption of "Filter For:" in the filter bar window.
exFilterBarFilterTitle	4	Defines the title for the filter tooltip.
exFilterBarPatternFilterTitle	5	Defines the title for the filter pattern tooltip.
exFilterBarTooltip	6	Defines the tooltip for filter window.
exFilterBarPatternTooltip	7	Defines the tooltip for filter pattern window
exFilterBarFilterForTooltip	8	Defines the tooltip for "Filter For:" window
exFilterBarIsBlank	9	Defines the caption of the function 'IsBlank' in the control's filter bar.
exFilterBarIsNonBlank	10	Defines the caption of the function 'not IsBlank' in the control's filter bar.
exFilterBarAnd	11	Customizes the ' and ' 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 delimitated by space characters. By default, the exFilterBarDateMonths is "January February March April May June July August September October November December".
exFilterBarDateWeekDays	18	Specifies the shortcut for the weekdays to be displayed in a date filter window. The list of shortcut for the weekdays should be separated by space characters. By default, the exFilterBarDateWeekDays is "S M T W T F S". The first shortcut in the list indicates the shortcut for the Sunday, the second shortcut indicates the shortcut for Monday, and so on.
exFilterBarChecked	19	Defines the caption of (Checked) in the filter bar window. The exFilterBarChecked option is displayed only if the FilterType property is exCheck. If the Description(exFilterBarChecked) property is empty, the (Checked) predefined item is not shown in the drop down filter window. If the user selects the (Checked) item the control filter checked items. The CellState property indicates the state of the cell's checkbox.

exFilterBarUnchecked	20	Defines the caption of (Unchecked) in the filter bar window. The exFilterBarUnchecked option is displayed only if the FilterType property is exCheck. If the Description(exFilterBarUnchecked) property is empty, the (Unchecked) predefined item is not shown in the drop down filter window. If the user selects the (Unchecked) item the control filter unchecked items. The CellState property indicates the state of the cell's checkbox.
exFilterBarIsChecked	21	Defines the caption of the 'IsChecked' function in the control's filter bar. The 'IsChecked' function may appear only if the user selects (Checked) item in the drop down filter window, when the FilterType property is exCheck
exFilterBarIsUnchecked	22	Defines the caption of the 'not IsChecked' function in the control's filter bar. The 'not IsChecked' function may appear only if the user selects (Unchecked) item in the drop down filter window, when the FilterType property is exCheck
exFilterBarOr	23	Customizes the 'or' operator in the control's filter bar when multiple columns are used to filter the items in the control.
exFilterBarNot	24	Customizes the 'not' operator in the control's filter bar.
exFilterBarExclude	25	Specifies the 'Exclude' caption being displayed in the drop down filter. The Exclude option is displayed in the drop down filter window only if the FilterList property includes the exShowExlcude flag.
exColumnsFloatBar	26	Specifies the caption to be shown on control's Columns float bar.

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	DoubleDottted line
ThinLine	5	Thin line
DoubleThinLine	6	Double thin line

constants exClipboardFormatEnum

Defines the clipboard format constants. Use GetFormat property to check whether the clipboard data is of given type

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

constants exOLEDragOverEnum

State transition constants for the OLEDragOver event

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

constants exOLEDropEffectEnum

Drop effect constants for OLE drag and drop events.

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

constants exOLEDropModeEnum

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

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

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

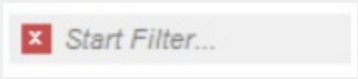
constants ExpandButtonEnum

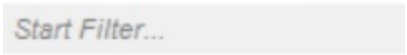
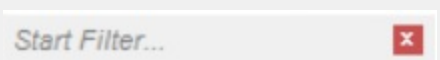
Defines how the control displays the expanding/collapsing buttons.

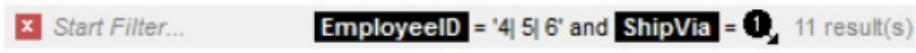
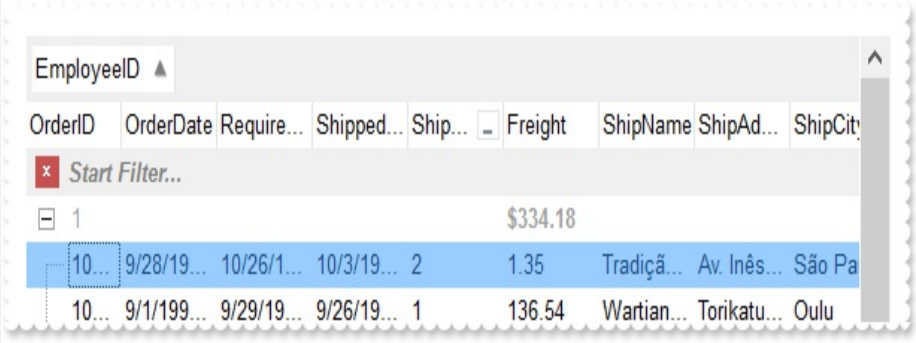
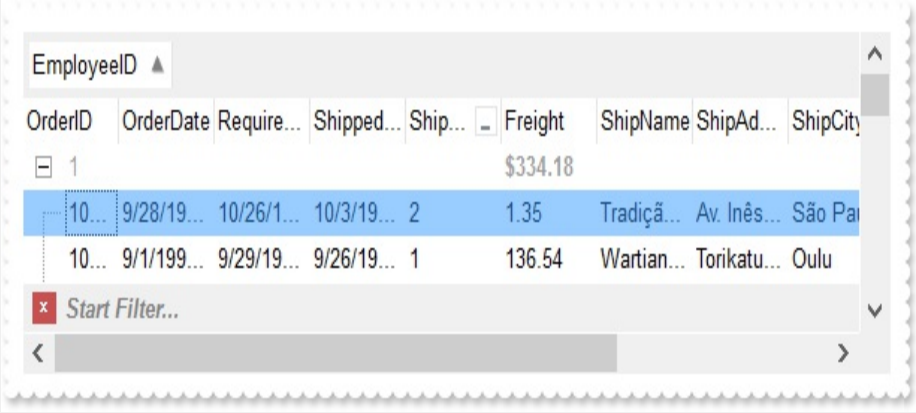
Name	Value	Description
exNoButtons	0	The control displays no expand buttons.
exPlus	-1	A plus sign is displayed for collapsed items, and a minus sign for expanded items.(⊞ ⊟)
exArrow	1	The control uses icons to display the expand buttons.(▸ ▾)
exCircle	2	The control uses icons to display the expand buttons. (⊕ ⊖)
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 though the list or control's filter bar using the ALT + Up/Down keys. If missing, the control's filter bar is always shown if any of the following flags is present exFilterBarPromptVisible, exFilterBarVisible, exFilterBarCaptionVisible.
exFilterBarShowCloseIfRequired	512	The exFilterBarShowCloseIfRequired flag indicates that the close button of the control's filter bar is displayed only if the control has any currently filter applied. The Background(exFooterFilterBarButton) property on -1 hides permanently the close button of the control's filter bar. 
exFilterBarShowCloseOnRight	1024	The exFilterBarShowCloseOnRight flag specifies that the close button of the control's filter bar should be displayed on the right side. If the control's RightToLeft property is True, the close button of the control's filter bar would be automatically displayed on the left side. 

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

constants 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:

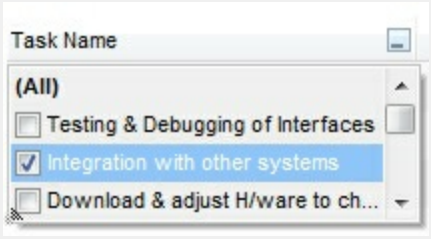
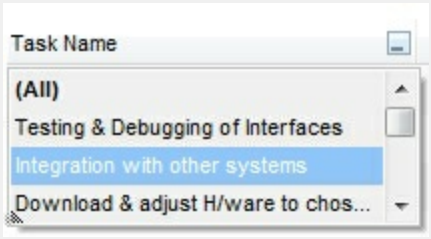
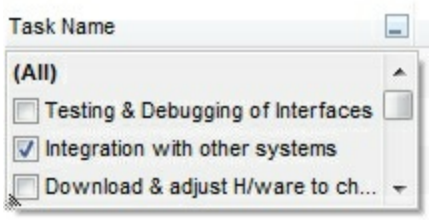
Name	Value	Description
exItemsWithoutChilds	0	Items (and parent-items) that match the filter are shown (no child-items are included)
exItemsWithChilds	1	Items (parent and child-items) that match the filter are shown
exRootsWithoutChilds	2	Only root-items (excludes child-items) that match the filter are 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.

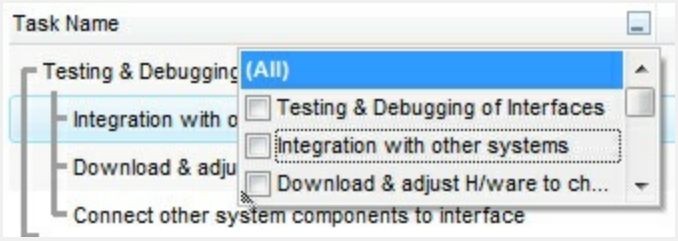
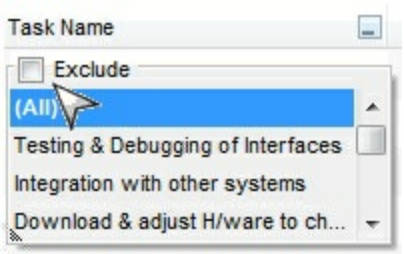
constants FilterListEnum

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 bellow:

Name	Value	Description
exAllItems	0	The filter's list includes all items in the column.
exVisibleItems	1	The filter's list includes only visible (filtered) items from the column. The visible items include child items of collapsed items.
exNoItems	2	The filter's list does not include any item from the column. Use this option if the drop down filter displays a calendar control for instance.
exLeafItems	3	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.
exIncludeInnerCells	64	The exIncludeInnerCells flag specifies whether the inner cells values are included in the drop down filter's list. The SplitCell method adds an inner cell, on in other words splits a cell.
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.
exShowCheckBox	256	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. The following screen shot shows the drop down filter **with** or **with no** exShowCheckBox flag:  or 
exHideCheckSelect	512	The selection background is not shown for checked items in the filter's list. This flag can be combined with exShowCheckBox. The following screen shot shows no selection background for the checked items: 
		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

exShowFocusItem	1024	item being focused in the drop down filter list is the one you have in the control's selection. This flag has effect also, if displaying a calendar control in the drop down filter list. The following screen shot shows the focused item in the filter's list (The Integration ... item in the background is the focused item, and the same is in the filter's list) : 
exShowPrevSelectOpaque	2048	By default, the previously selection in the drop down filter's list is shown using a semi-transparent color. Use this flag to show the previously selection using an opaque color. The exSelFilterForeColor and exSelFilterBackColor options defines the filter's list selection foreground and background colors.
exEnableToolTip	4096	This flag indicates whether the filter's tooltip is shown. The Description(exFilterBarTooltip,exFilterBarPatternTooltip ...) properties defines the filter's tooltips.
exShowExclude	8192	This flag indicates whether the Exclude option is shown in the drop down filter window. This option has effect also if the drop down filter window shows a calendar control. The following screen shot shows the Exclude field in the drop down filter window: 
exShowBlanks	16384	This flag indicates whether the (Blanks) and (NonBlanks) items are shown in the filter's list

constants FilterPromptEnum

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

The filter prompt feature supports the following values:

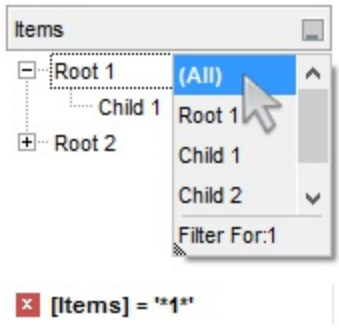
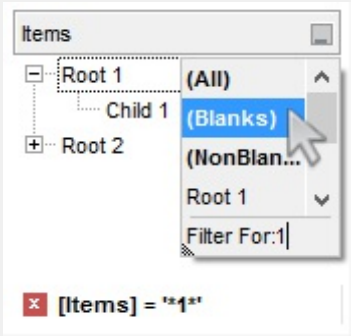
Name	Value	Description
exFilterPromptContainsAll	1	The list includes the items that contains all specified sequences in the filter. Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
exFilterPromptContainsAny	2	The list includes the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

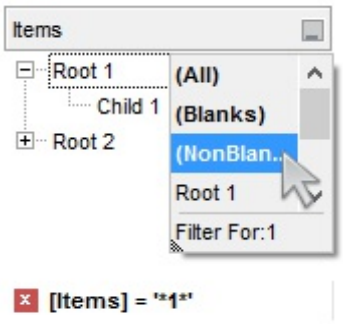
- ' ' space delimits the patterns inside the filter

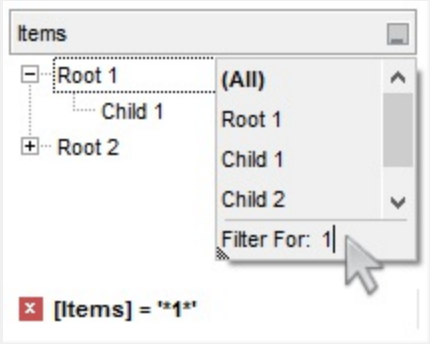
exFilterPromptCaseSensitive	256	Filtering the list is case sensitive. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith, exFilterPromptEndWith or exFilterPromptPattern.
exFilterPromptStartWords	4608	The list includes the items that starts with specified words, in any position. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.
exFilterPromptEndWords	8704	The list includes the items that ends with specified words, in any position. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.
exFilterPromptWords	12800	The filter indicates a list of words. Can be combined with exFilterPromptContainsAll, exFilterPromptContainsAny, exFilterPromptStartWith or exFilterPromptEndWith.

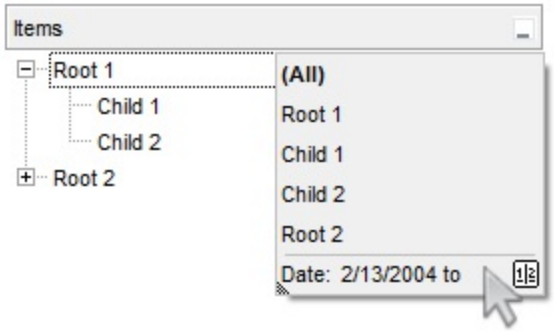
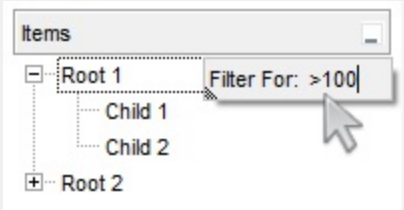
constants FilterTypeEnum

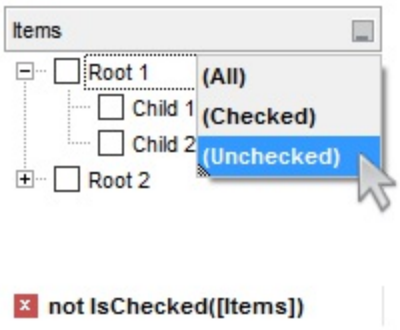
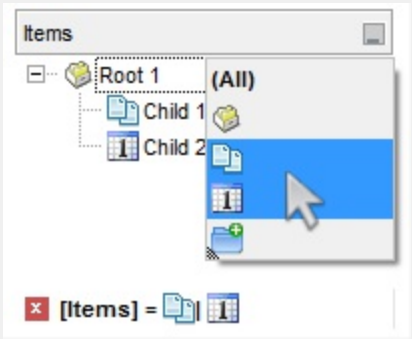
The FilterTypeEnum type defines the type of filter applies to a column. Use the FilterType property to specify the type of filter being used. Use the Filter property to specify the filter being used. The value for Filter property depends on the FilterType property. Use the Description property to customize the captions for control filter bar window. The FilterList property indicates the values the drop-down filter includes. The FilterTypeEnum type supports the following values:

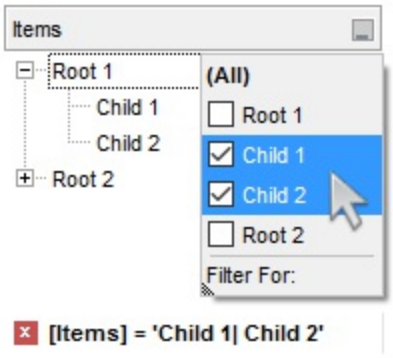
Name	Value	Description
exAll	0	No filter applied. Use the Description property to change the "(All)" caption in the drop down filter. 
exBlanks	1	Only blank items are included. Use the Description property to change the "(Blanks)" caption in the drop down filter. The Filter property has no effect. 
exNonBlanks	2	Only non blanks items are included. Use the Description property to change the "(NonBlanks) " caption in the drop down filter. The Filter property has no effect.

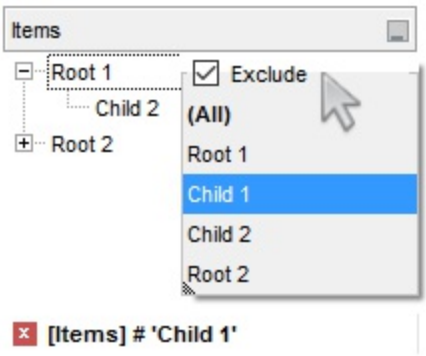


exPattern	3	Only items that match the pattern are included. The Filter property defines the pattern. A pattern may contain the wild card characters '?' for any single character, '*' for zero or more occurrences of any character, '#' for any digit character, and [chars] indicates a group of characters. If any of the *, ?, # or | characters are preceded by a \ (escape character) it masks the character itself. The Def(exFilterPatternTemplate) property specifies the template for the column's filter when the Filter property or the 'Filter For' field is populated. The exFilterDoCaseSensitive flag can be combined with exPattern or exFilter types, indicating that case-sensitive filtering should be performed. For instance: • "*1", only items that ends with 1 are included • "A*|B*", only items that starts with a/A or b/B 
		Only items (of date type) within the specified range are included. The Filter property 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 change the "to" conjunction used to split the dates in the interval. If the dateFrom value is missing, the control includes only

exDate	4	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. The DisplayFilterDate property specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for. For instance: • "2/13/2004 to" includes all items after 2/13/2004 inclusive • "2/13/2004 to Feb 14 2005" includes all items between 2/13/2004 and 2/14/2004 
exNumeric	5	Only items (of numeric type) within the specified range are included. The Filter property may include operators like <, <=, =, <>, >= or > and numbers to define rules to include numbers in the control's list. If the FilterType property is exNumeric, the drop down filter window doesn't display the filter list that includes items "(All)", "(Blanks)", ... and so on. For instance: • "100", filter items with the value 100 • "> 10 < 100", 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 Filter property on "0" filters for unchecked items, while "1" filters for checked items. A checked item has the the CellState property different than zero. An unchecked item has the CellState property on zero. For instance: • "0", only unchecked items are included • "1", only checked items are included 
exImage	10	Only items showing the specified icons (icon index) are included. The CellImage property indicates the cell's icon. Multiple icons are separated by the '|' character. The Filter property defines the list of icons, separated by the '|' character, to apply the filter. For instance: • "1", only items that displays the icons with the index 1 are included • "2|3", only items displaying the icons with index 2 or 3 are included 
		Only the items that are in the Filter property are

exFilter	240	included. Multiple items are separated by the '|' character. The exShowCheckBox flag of FilterList property displays a check box for each included item. The exFilterDoCaseSensitive flag can be combined with exPattern or exFilter types, indicating that case-sensitive filtering should be performed. For instance: • "Item 1", only items with the caption 'Item 1' are included • "Item 3|Item 3", only items displaying icons with an index of 2 or 3 are included 
exFilterDoCaseSensitive	256	If this flag is present, the column filtering is case-sensitive. If this flag is missing, the filtering is case-insensitive by default. The exFilterDoCaseSensitive flag can be used to enable case-sensitive filtering within the column. However, this flag is not applied to the filter prompt feature. The exFilterDoCaseSensitive flag can be combined with exPattern or exFilter types.
exFilterExclude	512	The flag indicates that the Exclude field of the column is checked, meaning items that match the filter are excluded from the list. The exShowExclude flag of FilterList property indicates whether the Exclude option is shown in the drop down filter window.


Items
Root 1
Child 2
Root 2
Exclude
(All)
Root 1
Child 1
Child 2
Root 2
[Items] # 'Child 1'

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 GridLinesStyleEnum

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

Name	Value	Description
exGridLinesDot	0	The control's gridlines are shown as dotted.
exGridLinesHDot4	1	The horizontal control's gridlines are shown as dotted.
exGridLinesVDot4	2	The vertical control's gridlines are shown as dotted.
exGridLinesDot4	3	The control's gridlines are shown as solid.
exGridLinesHDash	4	The horizontal control's gridlines are shown as dashed.
exGridLinesVDash	8	The vertical control's gridlines are shown as dashed.
exGridLinesDash	12	The control's gridlines are shown as dashed.
exGridLinesHSolid	16	The horizontal control's gridlines are shown as solid.
exGridLinesVSolid	32	The vertical control's gridlines are shown as solid.
exGridLinesSolid	48	The control's gridlines are shown as solid.
exGridLinesGeometric	512	exGridlinesGeometric. The control's gridlines are drawn using a geometric pen. 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 co?Ç?\

constants HierarchyLineEnum

Defines how the control paints the hierarchy lines. Use the TreeColumnIndex property to define the index of the column that displays the hierarchy. Use the LinesAtRoot property to connect root items. Use the HasLines property to connect a child items to their correspondent parent item.

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.

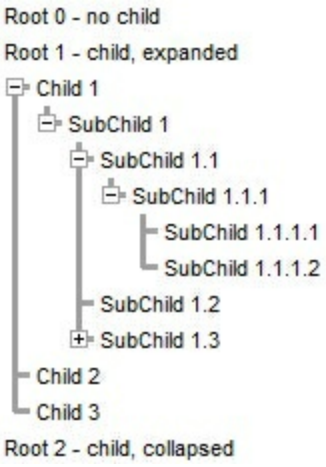
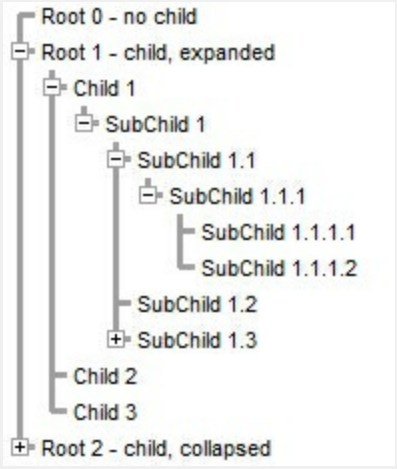
Name	Value	Description
exHTCell	0	In the cell's client area.
exHTExpandButton	1	In the +/- button associated with a cell. The HasButtons property specifies whether the cell displays a +/- sign to let user expands the item.
exHTCellIndent	2	In the indentation associated with a cell. The Indent property 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.
exHTCellCaptionIcon	1044	(HEXA 414) In the icon's area inside the cell's caption. The <img> tag inserts an icon inside the cell's caption. The <img> tag is valid only if the CellCaptionFormat property exHTML.
exHTBottomHalf	2048	(HEXA 800) The cursor is in the bottom half of the row. If this flag is not set, the cursor is in the top half of the row. This is an OR combination with the rest of predefined values. For instance, you can check if the cursor is in the bottom half of the row using HitTestCode AND 0x800
exHTBetween	4096	(HEXA 1000) The cursor is between two rows. This is an OR combination with the rest of predefined

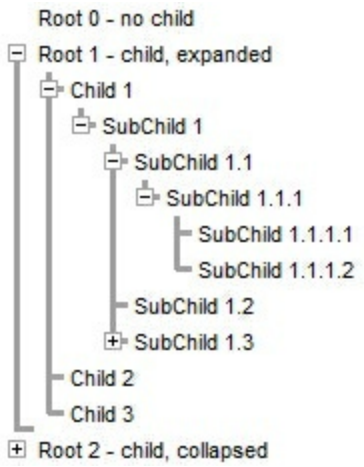
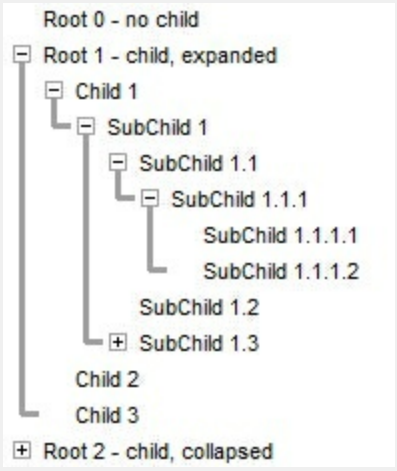
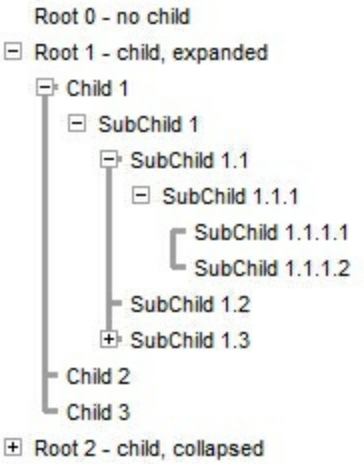
values. For instance, you can check if the cursor is between two items using HitTestCode AND 0x1000

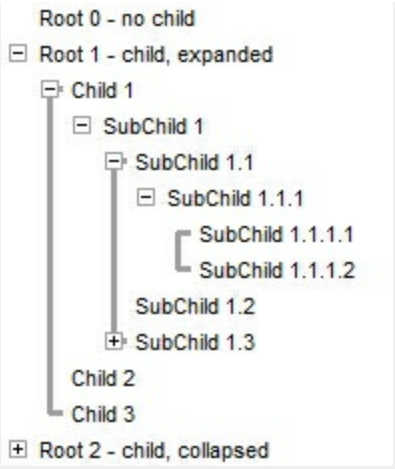
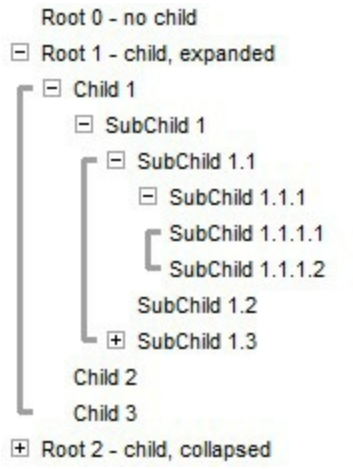
constants LinesAtRootEnum

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	
exGroupLines	2	The lines between root items are no shown, and the links show the items being included in the group. 
exGroupLinesInside	3	The lines between root items are no shown, and the links are shown between child only. 
		The lines between root items are no shown, and the links are shown for first and last visible child item.

exGroupLinesInsideLeaf	4	
exGroupLinesOutside	5	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. 

constants PictureDisplayEnum

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

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

constants ScrollBarEnum

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

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

constants ScrollBarsEnum

Specifies which scroll bars will be visible on a control.

Name	Value	Description
exNoScroll	0	No scroll bars are shown
exHorizontal	1	Only horizontal scroll bars are shown.
exVertical	2	Only vertical scroll bars are shown.
exBoth	3	Both horizontal and vertical scroll bars are shown.
exDisableNoHorizontal	5	The horizontal scroll bar is always shown, it is disabled if it is unnecessary.
exDisableNoVertical	10	The vertical scroll bar is always shown, it is disabled if it is unnecessary.
exDisableBoth	15	Both horizontal and vertical scroll bars are always shown, disabled if they are unnecessary.

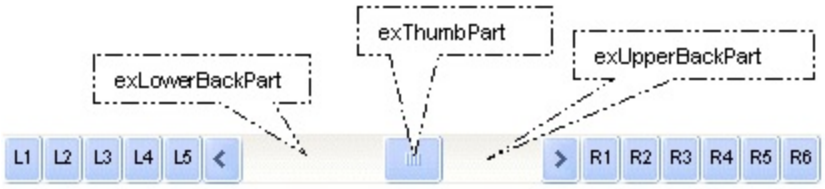
constants ScrollEnum

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

Name	Value	Description
exScrollUp	0	Scrolls up the control by a single line.
exScrollDown	1	Scrolls down the control by a single line.
exScrollVTo	2	Scrolls vertically the control to a specified position.
exScrollLeft	3	Scrolls the control to the left by a single pixel, or by a single column if the ContinueColumnScroll property is True.
exScrollRight	4	Scrolls the control to the right by a single pixel, or by a single column if the ContinueColumnScroll property is True.
exScrollHTo	5	Scrolls horizontaly the control to a specified position.

constants ScrollPartEnum

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



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

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

constants SortOnClickEnum

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

Name	Value	Description
exNoSort	0	The column is not sorted when the 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 icons, but it doesn't sort the column. The user is responsible with listing the items as being sorted. Use the ItemByPosition property to access the sorted column in their order.

constants SortOrderEnum

Specifies the column's sort order. Use the SortOrder property to specify the column's sort order.

Name	Value	Description
SortNone	0	The column is not sorted. (if the control supports sorting by multiple columns, the column is removed from the sorting columns collection)
SortAscending	1	The column is sorted ascending. (if the control supports sorting by multiple columns, the column is added to the sorting columns collection)
SortDescending	2	The column is sorted descending. (if the control supports sorting by multiple columns, the column is added to the sorting columns collection)

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 ItemsAllowSizingEnum

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

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

constants UIVisualThemeEnum

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

constants VAlignmentEnum

Specifies how the cell's caption is vertically aligned. Use the CellVAlignment property to align vertically the cell's caption.

Name	Value	Description
TopAlignment	0	The caption is aligned to top of the cell
MiddleAlignment	1	The cell's caption is vertically centered
BottomAlignment	2	The caption is aligned to bottom of the cell

Appearance object

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

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

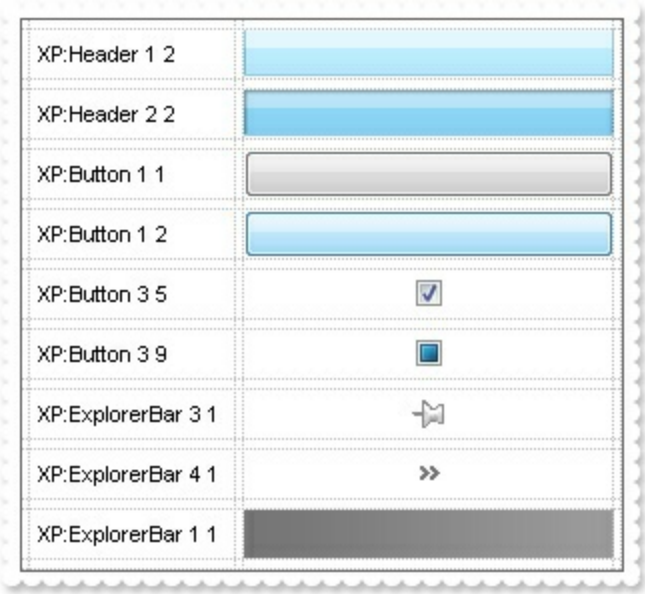
method Appearance.Add (ID as Long, Skin as Variant)

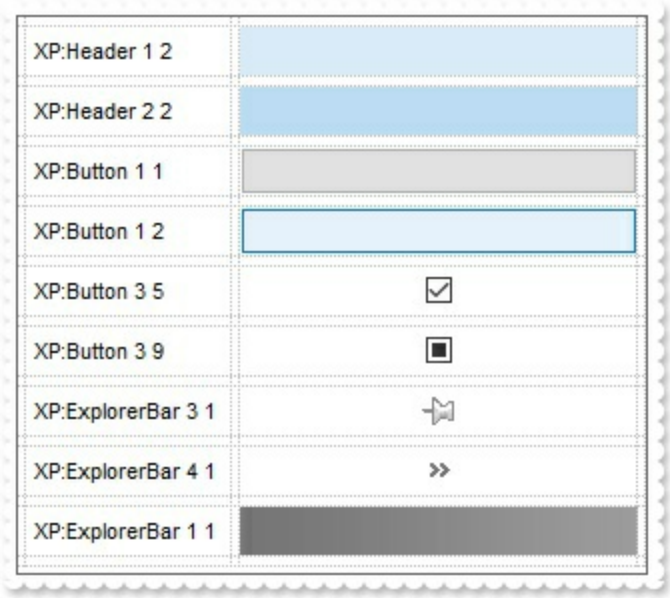
Adds or replaces a skin object to the control.

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

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

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



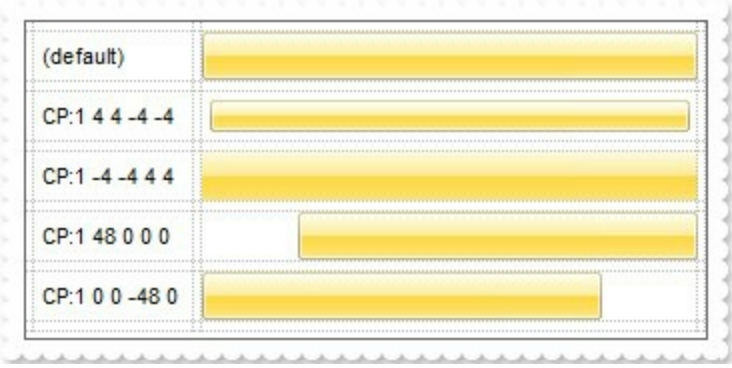


Skin as Variant

- 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:



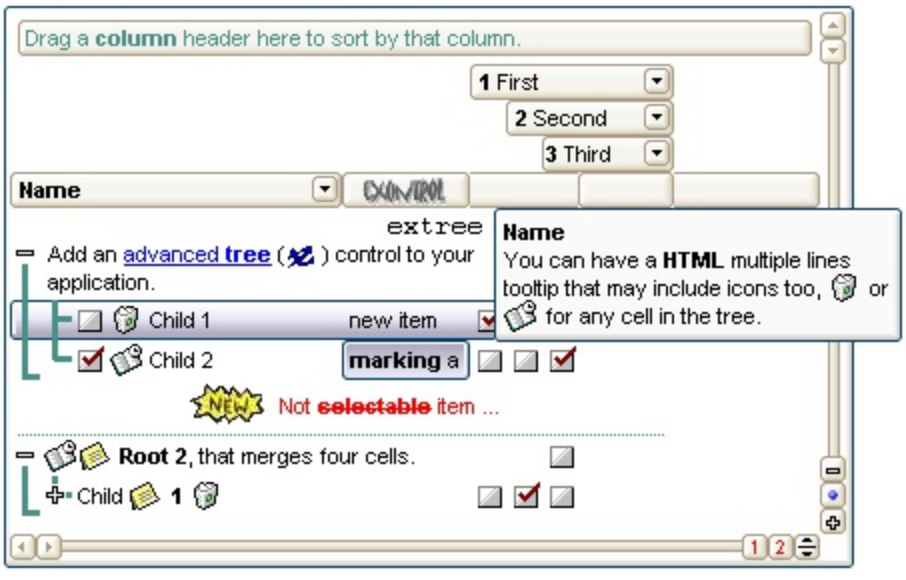
Return	Description
Boolean	A Boolean expression that indicates whether the new skin was added or replaced.

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.

(1)

The identifier you choose for the skin is very important to be used in the background properties like explained bellow. Shortly, the color properties uses 4 bytes (DWORD, double WORD, and so on) to hold a RGB value. More than that, the first byte (most significant byte in the color) is used only to specify system color. if the first bit in the byte is 1, the rest of bits indicates the index of the system color being used. So, we use the last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. So, since the 7 bits can cover 127 values, excluding 0, we have 126 possibilities to store an identifier in that byte. This way, a DWORD expression indicates the background color stored in RRGGBB format and the index of the skin (ID parameter) in the last 7 bits in the high significant byte of the color. For instance, the BackColor = BackColor Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the object that BackColor is applied.

(2)



(3)

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

- control's **borders** using the Appearance property
- control's **header bar**, BackColorHeader property
- control's **filter bar**, FilterBarBackColor property
- control's **sort bar**, BackColorSort property
- control's **scroll bar**, Background property
- the caption of the control's sort bar, BackColorSortCaption property
- **selected item** or cell, SelBackColor property
- **item**, ItemBackColor property
- **cell**, CellBackColor property
- cell's **button**, "**drop down**" filter bar button, "close" filter bar button, **tooltips**, **scrollbars,** and so on, Background property
- CellImage, CellImages, HeaderImage, CheckImage or RadioImage, HasButtonsCustom property
- Any HTML caption that includes an <img> tag.

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 Tree1
  With .VisualAppearance
    .Add &H23, App.Path + "\selected.ebn"
  End With
  .SelForeColor = RGB(0, 0, 0)
  .SelBackColor = .SelBackColor Or &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_tree.GetVisualAppearance().Add( 0x23,
COleVariant(_T("D:\\Temp\\ExTree_Help\\selected.ebn")) );
m_tree.SetSelBackColor( m_tree.GetSelBackColor() | 0x23000000 );
```

```
m_tree.SetSelForeColor( 0 );
```

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

```
With AxTree1
   With .VisualAppearance
      .Add(&H23, "D:\Temp\ExTree_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):

```
axTree1.VisualAppearance.Add(0x23, "D:\\Temp\\ExTree_Help\\selected.ebn");
axTree1.Template = "SelBackColor = 587202560";
```

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

```
With thisform.Tree1
   With .VisualAppearance
      .Add(35, "D:\Temp\ExTree_Help\selected.ebn")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = .SelBackColor + 587202560
EndWith
```

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

The first screen (1) shot was generated using the following template (On Windows XP):

```
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")
BackColorHeader = 16777216
SelBackColor = 33554432
Background(1) = 50331648
Background(0) = 67108864
Background(20) = 33554432
Background(21) = 1
SelForeColor = 0

MarkSearchColumn = False
ShowFocusRect = False
LinesAtRoot = -1
ConditionalFormats
{
   Add("%2 > 15")
   {
      Bold = True
      ForeColor = RGB(0,128,0)
      ApplyTo = 2
   }
   Add("%2 > 10 and %2 < 18")
   {
      Bold = True
      ForeColor = RGB(255,128,0)
      ApplyTo = 2
   }

}
Columns
{
   Add("A")
   {
      DisplayFilterButton = True
      Editor.EditType = 4
   }
   Add("B").Editor.EditType = 4
   Add("A+B").ComputedField = "%0 + %1"
```

```
}
Items
{
    Dim h, h1
    h = InsertItem(,,"Group 1")
    CellEditorVisible(h,0) = False
    CellEditorVisible(h,1) = False
    CellCaptionFormat(h,2) = 1
    h1 = InsertItem(h,,16)
    CellCaption(h1,1) = 17
    h1 = InsertItem(h,,2)
    CellCaption(h1,1) = 11
    h1 = InsertItem(h,,2)
    CellCaption(h1,1) = 9
    ExpandItem(h) = True
    h = InsertItem(,,"Group 2")
    CellEditorVisible(h,0) = False
    CellEditorVisible(h,1) = False
    CellCaptionFormat(h,2) = 1
    h1 = InsertItem(h,,16)
    CellCaption(h1,1) = 9
    h1 = InsertItem(h,,12)
    CellCaption(h1,1) = 11
    h1 = InsertItem(h,,2)
    CellCaption(h1,1) = 2
    ExpandItem(h) = True
    SelectItem(h) = True
}
EndUpdate
```

The second screen (2) shot was generated using the following template:

```
BeginUpdate

Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl

Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl
```

```
Images("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0pIUrlkt

Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl


VisualAppearance
{
   ' Header

Add(1,"gBFLBCJwBAEHhEJAEGg4BawDg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAGEYxR

   ' HeaderFilterBarButton

Add(2,"gBFLBCJwBAEHhEJAEGg4BAQEg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAGEYxR


Add(3,"gBFLBCJwBAEHhEJAEGg4BBAEg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAGEYxRg

   ' SelectedItem
   Add(4,
"gBFLBCJwBAEHhEJAEGg4BV4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK

   ' Mark a cell

Add(5,"gBFLBCJwBAEHhEJAEGg4BEcMQAAYAQGKIYBkAKBQAGaAoDDMOILQiMQxDPBMK

}
BackColorHeader = 16777216          '0x01BBGGRR
BackColorSortBarCaption = 33488896     '0x01BBGGRR
FilterBarBackColor = 16777216        '0x01BBGGRR
Background(0) = 33554432             '0x02BBGGRR
Background(1) = 50331648             '0x03BBGGRR
SelBackColor = 67108864             '0x04BBGGRR
BackColorSortBar = RGB(61,101,183)
FilterBarForeColor = RGB(255,255,255)
ForeColorHeader = RGB(255,255,255)
```

```
ForeColorSortBar = RGB(255,255,255)
SelForeColor = 0
DefaultItemHeight = 20
HeaderHeight = 20
SortBarHeight = 20
DrawGridLines = -1
ShowFocusRect = False
SortBarVisible = True
MarkSearchColumn = False
LinesAtRoot = 1
BackColor = RGB(255,255,255)
BackColorLevelHeader = RGB(255,255,255)
ScrollBySingleLine = True
HasLines = 2
HasButtons = 3
CheckImage(1) = 4
CheckImage(0) = 5
CheckImage(2) = 6
Columns
{
   " Name "
   {
      HeaderBold = True
      DisplayFilterButton = True
      DisplayFilterDate = True
      Width = 196
   }
   " Value"
   {
      DisplayFilterButton = True
      FilterBarDropDownWidth = -100
   }
   1
   {
      AllowSizing = False
      HTMLCaption = " 1 First"
      Def(0) = True
```

```
    LevelKey = 1
    Width = 25
    Alignment = 1
  }
  2
  {
    AllowSizing = False
    HTMLCaption = " 2 Second"
    Def(0) = True
    LevelKey = 1
    Width = 25
    Alignment = 1
  }
  3
  {
    AllowSizing = False
    HTMLCaption = "3 Third"
    Def(0) = True
    LevelKey = 1
    Width = 25
    PartialCheck = True
    Alignment = 1
  }
  ""
  {
    LevelKey = 1
    Width = 20
  }

}
Items
{
  Dim h, h1,hx
  h = AddItem("exTreeAdd an advanced tree control to your application.")
  CellTooltip(h,0) = "You can have a HTML multiple lines tooltip for any cell in the tree."
  CellCaptionFormat(h,0) = 1
  CellSingleLine(h,0) = False
```

```
    CellImages(h,1) = "1,2,3,"
    CellHAlignment(h,1) = 2
    CellMerge(h,0) = 1
    CellMerge(h,0) = 2
    CellMerge(h,0) = 3

    h1 = InsertItem(h,,"Child 1")
    CellHasCheckBox(h1,0) = True
    CellCaption(h1,1) = "new item"
    CellImage(h1,0) = 1
    SelectItem(h1) = True

    h1 = InsertItem(h,,"Child 2")
    CellCaption(h1,1) = "marking a cell"
    CellHasCheckBox(h1,0) = True
    CellState(h1,0) = 1
    CellCaptionFormat(h1,1) = 1
    CellImage(h1,0) = 2
    CellBackColor(h1,1) = 83886080    '0x5RRGGBB

    ExpandItem(h) = True

    h = AddItem("")
    CellCaption(h,1) = "Non selectable item ..."
    CellCaptionFormat(h,1) = 1
    ItemDivider(h) = 1
    ItemHeight(h) = 28
    ItemDividerLine(h) = 3
    CellHAlignment(h,1) = 1
    SelectableItem(h) = False
    CellPicture(h,1) =
"gBHJJGHA5MIqAAXAD3AENhozhpmhqZhrMhr/h0QGcQM0QTMQZkQf8QAESGcSM0STM



    h = AddItem("Root 2")
    CellImages(h,0) = "2,3"
    ItemBold(h) = True
```

```
   CellMerge(h,0) = 1
   CellMerge(h,0) = 2
   CellMerge(h,0) = 3

   h1 = InsertItem(h,,"Child 1")
   hx = InsertControlItem(,"MSCAL.Calendar")
   ItemHeight(hx) = 164
   ItemObject(hx)
   {
      BackColor = RGB(250,250,250)
      DayLength = 0
   }
   SelectableItem(hx) = False
}

EndUpdate
```

The third screen (3) shot was generated using the following template:

```
BeginUpdate

Images("gBJJgBAIHAAJAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaGEaAIAEEbjMjIErlktl0

Images("gBJJgBAIEAAGAEGCAAhb/hz/EIAh8Tf5CJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0



HTMLPicture("logo") =
"gBHJJGHA5MJXAAfAD3AENhoBiACiQDigEiwFjAHjQIjgJjwKkALkQMkgNkwOlAPlQQIgRlwS



VisualAppearance
{
   ' Header
   Add(1,
"gBFLBCJwBAEHhEJAEGg4BcoDg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK

   ' HeaderFilterBarButton
```

```
Add(2,"gBFLBCJwBAEHhEJAEGg4BCwEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

Add(3,"gBFLBCJwBAEHhEJAEGg4BFQEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

    ' SelectedItem
    Add(4,
"gBFLBCJwBAEHhEJAEGg4BV4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK

Add(5,"gBFLBCJwBAEHhEJAEGg4Ba4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAga

Add(6,"gBFLBCJwBAEHhEJAEGg4BaAFg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

Add(7,"gBFLBCJwBAEHhEJAEGg4BYIEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAI

    'Spin
    Add(8,
"gBFLBCJwBAEHhEJAEGg4BDAGg6AADACAxRDAMgBQKAAzAFBIYhxASCBhGaCYUACCIVRI

Add(9,"gBFLBCJwBAEHhEJAEGg4BIAGg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA

    Add(10,
"gBFLBCJwBAEHhEJAEGg4BDIGg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAIIRZGM
)
    Add(11,
"gBFLBCJwBAEHhEJAEGg4BHwGg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAIIRZG
```

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

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

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

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

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

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

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

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

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

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

WP_HELPBUTTON = 23	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_HORZSCROLL = 25	HSS_NORMAL = 1 = 2 HSS_PUSHED HSS_DISABLED =
WP_HORZTHUMB = 26	HTS_NORMAL = 1 2 HTS_PUSHED = 3 HTS_DISABLED = 4
WP_MAX_BUTTON	MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_MAXCAPTION = 5	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
WP_MDICLOSEBUTTON = 20	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
WP_MDIHELPBUTTON = 24	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_MDIMINBUTTON = 16	MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MDIRESTOREBUTTON = 22	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_MDISYSBUTTON = 14	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_MINBUTTON = 15	MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MINCAPTION = 3	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_RESTOREBUTTON = 21	RBS_NORMAL = 1 = 2 RBS_PUSHED

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

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

method Appearance.Clear ()

Removes all skins in the control.

Type	Description

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

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

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

method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

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

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

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

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

property Appearance.RenderType as Long

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

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

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

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

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

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

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

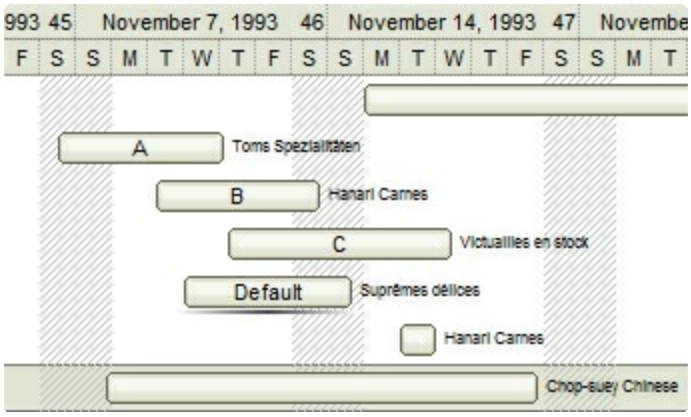
In the following screen shot the following objects displays the current EBN with a different color:

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

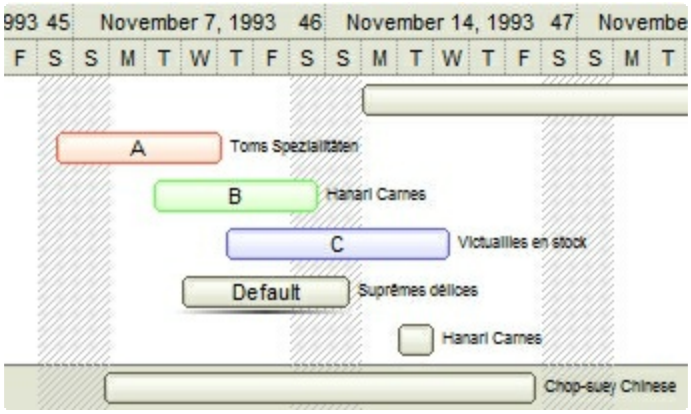
- "C" in Blue (RGB(0,0,255), for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

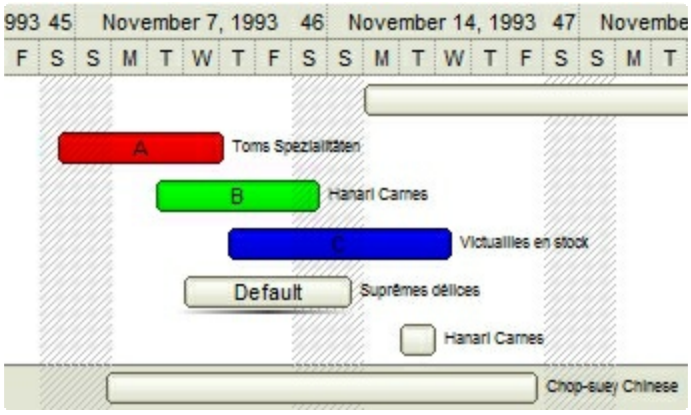
- ***-3**, no color is applied*. For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.



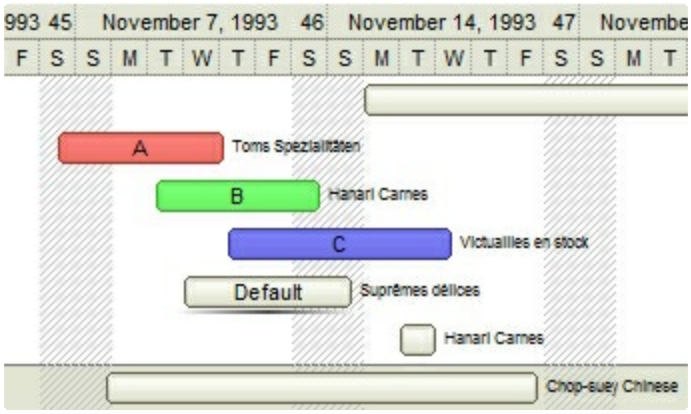
- ***-2**, OR-color scheme*. The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the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



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

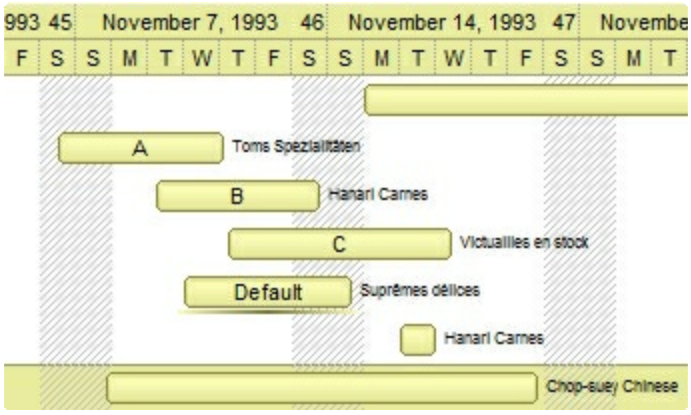


- ***0**, default*, the specified color is applied to the EBN. For instance, the BackColorHeader = &H1FF0000, applies a Blue color to the object. This option could be used to specify any color for the part of the components, that support EBN objects, not only solid colors.

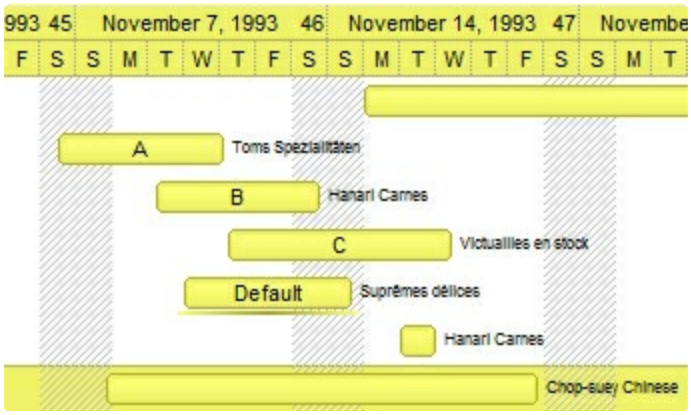


- **0xAABBGGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on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

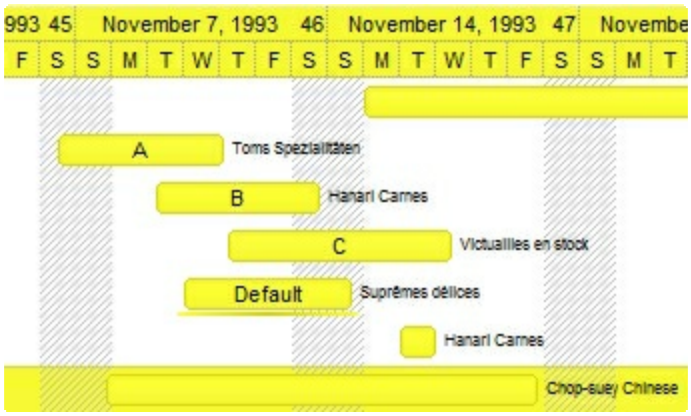
*The following picture shows the control with the RenderType property on 0x**40**00FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256):*



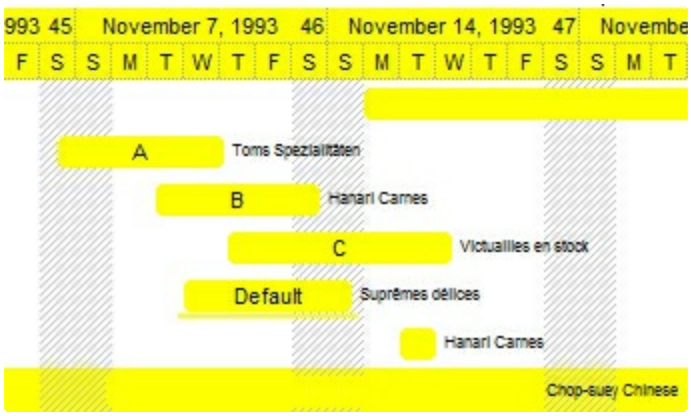
*The following picture shows the control with the RenderType property on 0x**80**00FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256):*



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



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


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

Column object

The ExTree component supports multiple columns. The Columns object contains a collection of Column objects. By default, the control doesn't add any default column, so the user has to add at least one column, before inserting any new items. The Column object holds information about a 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 caption into the column's header.
AllowDragging	Retrieves or sets a value indicating whether the user will be able to drag the column.
AllowSizing	Retrieves or sets a value indicating whether the user will be able to change the width of the visible columns by dragging.
AllowSort	Returns or sets a value that indicates whether the user can sort the column by clicking the column's header.
AutoSearch	Specifies the kind of searching while user types characters within the columns.
AutoWidth	Computes the column's width required to fit the entire column's content.
Caption	Retrieves or sets the text displayed to the column's header.
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.
DisplayExpandButton	Shows or hides the expanding/collapsing button in the column's header.
	Specifies whether the column's header displays the filter

DisplayFilterButton	button.
DisplayFilterDate	Specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for.
DisplayFilterPattern	Specifies whether the dropdown filter bar contains a textbox for editing the filter as pattern.
DisplaySortIcon	Retrieves or sets a value indicating whether the sort icon is visible on column's header, while the column is sorted.
Enabled	Returns or sets a value that determines whether a column's header can respond to user-generated events.
ExpandColumns	Specifies the list of columns to be shown when the current column is expanded.
Expanded	Expands or collapses the column.
Filter	Specifies the column's filter when filter type is exFilter, exPattern or exDate.
FilterBarDropDownWidth	Specifies the width of the drop down filter window proportionally with the width of the column.
FilterList	Specifies whether the drop down filter list includes visible or all items.
FilterOnType	Filters the column as user types characters in the drop down filter window.
FilterType	Specifies the column's filter type.
FireFormatColumn	Retrieves or sets a value that indicates whether the control fires FormatColumn to format the caption of a cell hosted by 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	Retrieves or sets a value indicating the index of an Image in the Images collection, which is displayed to 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 should appear in underline..
HeaderVertical	Specifies whether the column's header is vertically displayed.
HTMLCaption	Retrieves or sets the text in HTML format displayed in the column's header.
Index	Returns a value that represents the index of an object in a collection.
Key	Retrieves or sets 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.
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 caption into the column's header.

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

Use the Alignment property to change the column's alignment. Use the HeaderAlignment property to align the column's caption inside the column's header. By default, all columns are aligned to left. If the column displays the hierarchy lines, and if the Alignment property is RightAlignment the hierarchy lines are painted from right to left side. Use the HasLines property to display the control's hierarchy lines. Use the CellHAlignment property to align a particular cell.

property Column.AllowDragging as Boolean

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

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

Use the AllowDragging property to forbid user to change the column's position by dragging. If the AllowDragging is false, the column's position cannot be changed by dragging it to another position. Use the AllowSort property to avoid sorting a column when the user clicks the column's header. Use the AllowSizing property to allow user resizes a column at runtime.

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 ColumnAutoResize property of the control to fit the columns to the control's client area. Use the AllowSort property to avoid sorting a column when the user clicks the column's header. Use the AllowDragging property to forbid user to change the column's position by dragging. Use the Width property to specify the column's width.

property Column.AllowSort as Boolean

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

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

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

property Column.AutoSearch as AutoSearchEnum

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

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

By default, the AutoSearch property is exStartWith. The AutoSearch property has effect only if the AutoSearch property of the control is True. Use the AutoSearch property to define a 'contains' incremental search. If the AutoSearch property is exContains, the control searches for items that contains the typed characters. 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 width of the column to fit the entire column's content.

Use the AutoWidth property to arrange the columns to fit the entire control's content. The AutoWidth property 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 EXTREELibCtl.Tree)
   t.BeginUpdate
      Dim c As Column
      For Each c In t.Columns
         c.Width = c.AutoWidth
      Next
      t.EndUpdate
      t.Refresh
End Sub
```

The following C++ sample resizes all visible columns:

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

```
}
```

The following VB.NET sample resizes all visible columns:

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

The following C# sample resizes all visible columns:

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

The following VFP sample resizes all visible columns:

```
with thisform.Tree1
   .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.

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

property Column.ComputedField as String

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

Type	Description
String	A String expression that indicates the formula to compute the field/cell. The formula is applied to all cells in the column with the 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 FormatColumn property to format the column.

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 may be a combination of variables, constants, strings, dates and operators. 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 expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ...{any} specifies the value of the cell in the column with the index 0, 1 2, ... The CellCaption property defines the cell's value. For example, "%0 format ``" formats the value in the cell at index 0 using the current regional settings, while "int(%1)" converts the value in the cell at index 1 to an integer.*
- ***%C0**, **%C1**, **%C2**, ...{string} specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property gets the cell's formatted caption. The cell's displayed string may differ from its actual value.*

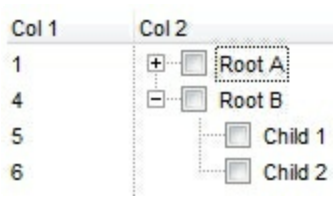
For example, if a cell displays HTML content, %0 returns the HTML format including the tags, while %C0 returns the cell's content as a plain string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell at index 1 to uppercase, while "%C0 left 2" returns the leftmost two characters of the caption in the cell at index 0.

- ***%CD0**, **%CD1**, **%CD2**, ...{any} specifies the cell's extra data in the column with the index 0, 1 2, ... The CellData property associates any extra/user data to a cell. For example, "%CD0 = your user data" specifies all cells in the column with index 0 whose CellData property is equal to your user data.*
- ***%CS0**, **%CS1**, **%CS2**, ...{number} specifies the cell's state in the column with the index 0, 1 2, ... The CellState property defines the state of a cell, indicating whether it is checked or unchecked. For example, "%CS0" identifies all checked items in the column with index 0, while "not %CS1" identifies all unchecked items in the column with index 1.*
- ***%CC0**, **%CC1**, **%CC2**, ... {number} retrieve the number of child items (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ChildCount property returns the number of child items. For example, "%CC0" identifies all parent items, while "%CC0 = 0" identifies all leaf items.*
- ***%CX0**, **%CX1**, **%CX2**, ... {boolean} returns true if the item hosting the cell is expanded, or false if it is collapsed (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ExpandItem property specifically indicates whether the item is expanded or collapsed. For example, "%CX0" refers to all expanded items, while "not %CX0" identifies all collapsed items*

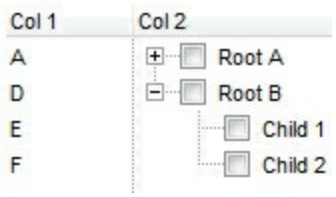
The predefined operators for auto-numbering are:

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

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

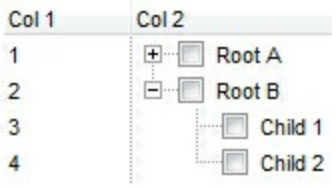


In the following screen shot the FormatColumn("Col 1") = "1 index 'A-Z'"

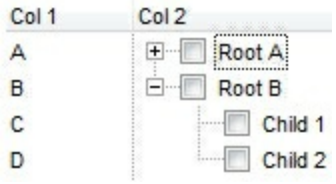


- number **apos** 'format' indicates the absolute position of the item. The first displayed item has the absolute position 0 (scrolling position on top), the next visible item is 1, and so on. The number indicates the starting position, while the format is a set of characters to be used for specifying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

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

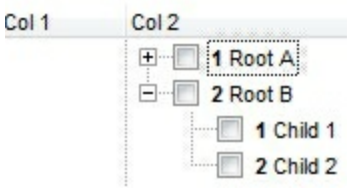


  In the following screen shot the FormatColumn("Col 1") = "1 apos 'A-Z'"

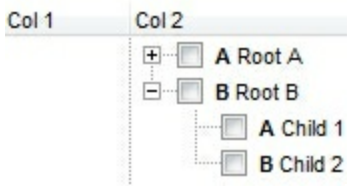


- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance,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

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

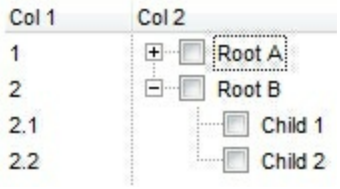


In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos 'A-Z' + ' ' + value"

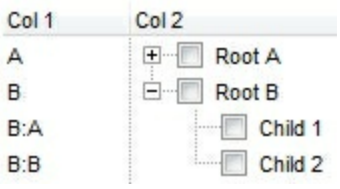


- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position.For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100. If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied, the opos gets the position of the item in the list of unfiltered items, while the pos gets the position of the item in the filtered list.*
- number **rpos** 'format' indicates the relative recursive position of the item. The recursive position indicates the position of the parent items too. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is of the following type "delimiter|format|format|...". If the format is missing, the delimiter is . character, and the positions are formatted as numbers. The format is applied consecutively to each parent item, from root to item itself.

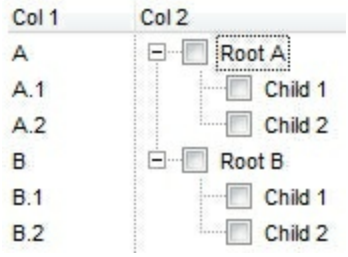
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ""



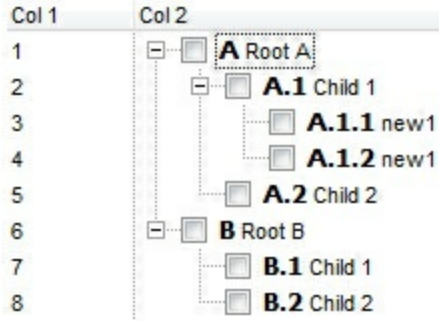
In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"



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



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



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

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

Samples:

1. "**1**", the cell displays 1
2. "**%0 + %1**", the cell displays the sum between cells in the first and second columns.
3. "**%0 + %1 - %2**", the cell displays the sum between cells in the first and second columns minus the third column.
4. "**(%0 + %1)*0.19**", the cell displays the sum between cells in the first and second columns multiplied with 0.19.
5. "**(%0 + %1 + %2)/3**", the cell displays the arithmetic average for the first three columns.
6. "**%0 + %1 < %2 + %3**", displays 1 if the sum between cells in the first two columns is less than the sum of third and forth columns.
7. "**proper(%0)**" formats the cells by capitalizing first letter in each word
8. "**currency(%1)**" displays the second column as currency using the format in the control panel for money
9. "**len(%0) ? currency(dbl(%0)) : ""** displays the currency only for not empty/blank cells.

10. "**int(date(%1)-date(%2)) + 'D ' + round(24*(date(%1)-date(%2) - floor(date(%1)-date(%2)))) + 'H'"** displays interval between two dates in days and hours, as xD yH
11. **"2:=((1:=int(0:= date(%1)-date(%0))) = 0 ? '' : str(=:1) + ' day(s)') + (3:=round(24*(=:0-floor(=:0))) ? (len(=:2) ? ' and ' : '') + =:3 + ' hour(s)' : '')"** displays the interval between two dates, as x day(s) [and y hour(s)], where the x indictaes the number of days, and y the number of hours. The hour part is missing, if 0 hours is displayed, or nothing is displayed if dates are identical.

property Column.CustomFilter as String

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

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

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

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

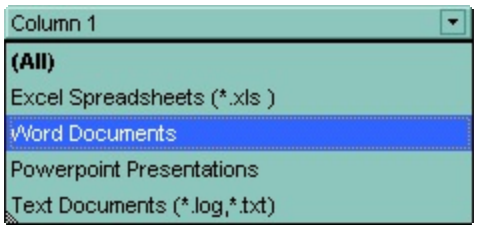
For example, you may have a list of documents and instead of listing the name of each document in the filter drop down list for the names column you may want to list the following:

Excel Spreadsheets
Word Documents
Powerpoint Presentations
Text Documents

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

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

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



property Column.Data as Variant

Associates an extra data to the column.

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

Use the Data property to assign any extra data to a column. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item. Use the SortUserData or SortUserDataString type to sort the column based on the CellData value.

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:

```
Tree1.Columns(0).Def(exCellHasCheckBox) = True
```

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

```
Tree1.Columns(0).Def(exCellBackColor) = RGB(240, 240, 240)
```

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

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

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

```
With AxTree1.Columns(0)
   .Def(EXTREELib.DefColumnEnum.exCellHasCheckBox) = True
End With
```

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

```
With AxTree1.Columns(0)
   .Def(EXTREELib.DefColumnEnum.exCellBackColor) = ToUInt32(Color.WhiteSmoke)
End With
```

where the ToUInt32 function converts a Color expression to OLE_COLOR,

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

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

```
axTree1.Columns[0].set_Def( EXTREELib.DefColumnEnum.exCellHasCheckBox, true );
```

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

```
axTree1.Columns[0].set_Def(EXTREELib.DefColumnEnum.exCellBackColor,
ToUInt32(Color.WhiteSmoke));
```

where the ToUInt32 function converts a Color expression to OLE_COLOR,

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

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

```
with thisform.Tree1.Columns(0)
   .Def( 0 ) = .t.
endwith
```

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

```
with thisform.Tree1.Columns(0)
    .Def( 4 ) = RGB(240,240,240)
endwith
```

property Column.DefaultSortOrder as Boolean

Specifies whether the default sort order is ascending or descending.

Type	Description
Boolean	A boolean expression that specifies whether the default sort order is ascending or descending. True means ascending, False means descending.

By default, the DefaultSortOrder property is False. Use the SortOnClick property to specify the operation that control should execute when the user clicks the column's header. Use the DefaultSortOrder to specify how the column is sorted at the first click on its header. Use the SortOrder property to sort a column. Use the SingleSort property to allow sorting by multiple columns.

property Column.DisplayExpandButton as Boolean

Shows or hides the expanding/collapsing button in the column's header.

Type	Description
Boolean	A Boolean expression that specifies whether the +/- expanding/collapsing button is shown in the column's header.

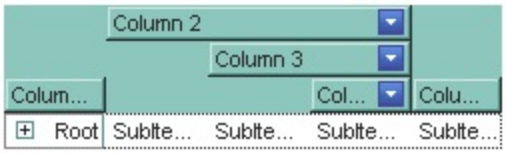
By default, the DisplayExpandButton property is True. The DisplayExpandButton property indicates whether the +/- expanding/collapsing button is shown in the column's header. Use the Expanded property to programmatically expand/collapse the columns. For instance, the Expanded property on False, collapse the column, while the Expanded property on True, expands the columns indicated by the ExpandColumns property. The ExpandColumns property specifies the columns to be shown/hidden when a column is expanded or collapsed.

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.

The column's filter button is displayed on the column's caption. The DisplayFilterPattern property determines whether the column's filter window includes the pattern field. Use the DisplayFilterDate property to include a date selector to the column's drop down filter window. Use the FilterBarDropDownHeight to specify the height of the drop down filter window. Use the FilterBarDropDownWidth property to specify the width of the drop down filter window. Use the FilterBarHeight property to specify the height of the filter bar header. Use the FilterList property to specify the list of items being included in the column's drop down filter list. Use the Background(exHeaderFilterBarButton) property to change the visual appearance for the drop down 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 specify the filter criteria using OR, AND or NOT operators. Use the CustomFilter property to define you custom filters. Use the ShowFilter method to show programmatically the column's drop down filter window. The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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.



The following VB sample changes the visual appearance for the "drop down" filter button. The sample applies the skin "" to the "drop down" filter buttons:

```
With Tree1
   With .VisualAppearance
      .Add &H1, App.Path + "\fbardd.ebn"
   End With
   .Background(exHeaderFilterBarButton) = &H1000000
End With
```

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

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x01,
COleVariant(_T("D:\\Temp\\ExTree.Help\\fbardd.ebn")) );
m_tree.SetBackground( 0 /*exHeaderFilterBarButton*/, 0x1000000 );
```

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

```
With AxTree1
   With .VisualAppearance
      .Add(&H1, "D:\Temp\ExTree.Help\fbardd.ebn")
   End With
   .set_Background(EXTREELib.BackgroundPartEnum.exHeaderFilterBarButton,
&H1000000)
End With
```

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

```
axTree1.VisualAppearance.Add(0x1, "D:\\Temp\\ExTree.Help\\fbardd.ebn");
axTree1.set_Background(EXTREELib.BackgroundPartEnum.exHeaderFilterBarButton,
0x1000000);
```

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

```
With thisform.Tree1
   With .VisualAppearance
      .Add(1, "D:\Temp\ExTree.Help\fbardd.ebn")
   EndWith
   .Object.Background(0) = 16777216
EndWith
```

The 16777216 value is the 0x1000000 value in hexadecimal.

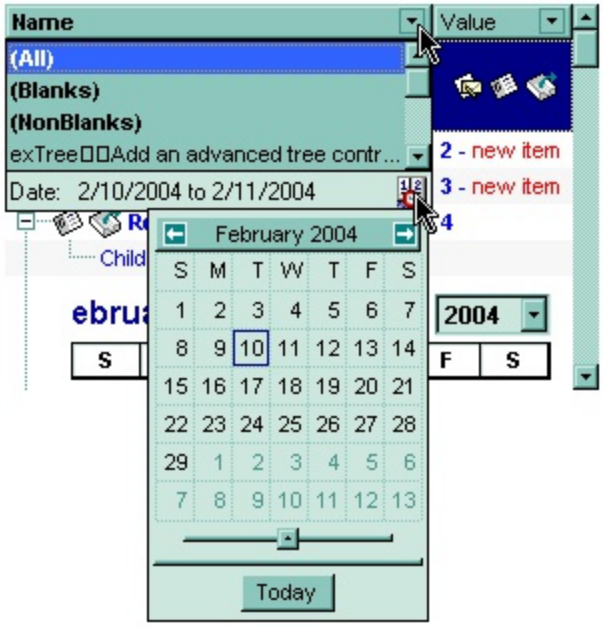
property Column.DisplayFilterDate as Boolean

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

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

By default, the DisplayFilterDate property is False. Use the DisplayFilterDate property to filter items that match a given interval of dates. *Use the FilterOnType property to enable the Filter-On-Type feature, that allows you to filter the control's data based on the characters you type.*

The 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. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter.



property Column.DisplayFilterPattern as Boolean

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

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

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

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

property Column.DisplaySortIcon as Boolean

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

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

Use the DisplaySortIcon property to hide the sort icon. Use the SortChildren property of the Items object to sort a column. Use the SortOrder property to sort a column. Use the SingleSort property to allow multiple sort columns.

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 in the column are disabled, no matter if the CellEnabled property is True. Use the Enabled property to disable the control.

property Column.ExpandColumns as String

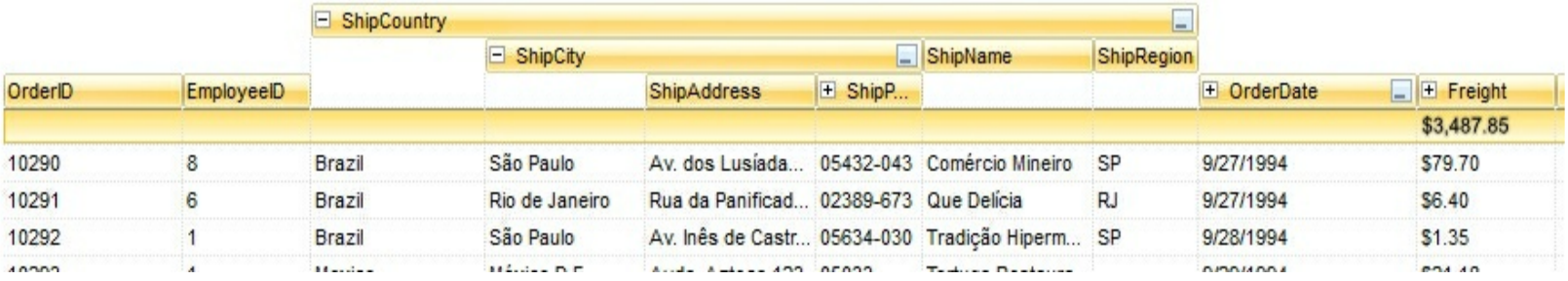
Specifies the list of columns to be shown when the current column is expanded.

Type	Description
String	A String expression that specifies the columns to be expanded/collapsed by current column. The expression contains the index of the columns to be shown or hidden, separated by comma. The list can includes the index of the current column, and so the column is always visible no matter if the column is expanded or collapsed.

By default, the ExpandColumns property is "". The ExpandColumns property specifies the columns to be shown/hidden when a column is expanded or collapsed. The ExpandColumns property can include the index of the current column, which indicates that the column is visible no matter if the column is expanded or collapsed. In other words, the Expanded/ExpandColumns properties provides expandable header. The Index property specifies the index of the column. The Expanded property specifies whether a column is expanded or collapsed. The DisplayExpandButton property indicates whether the +/- expanding/collapsing button is shown in the column's header. The HasButtons property specifies how the +/- buttons are shown.

The following screen shot shows the control's header when all columns are collapsed:

The following screen shot shows the control's header with columns expanded/collapsed :



property Column.Expanded as Boolean

Expands or collapses the column.

Type	Description
Boolean	A Boolean expression that specifies whether the column is expanded / collapsed.

By default, the Expanded property is True. Use the Expanded property to programmatically expand/collapse the columns. For instance, the Expanded property on False, collapse the column, while the Expanded property on True, expands the columns indicated by the ExpandColumns property. The ExpandColumns property specifies the columns to be shown/hidden when a column is expanded or collapsed. The DisplayExpandButton property indicates whether the +/- expanding/collapsing button is shown in the column's header.

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. 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 FilterCriteria property to specify the filter criteria using OR, AND or NOT operators. Use the ShowFilter method to show programmatically the column's drop down filter window.

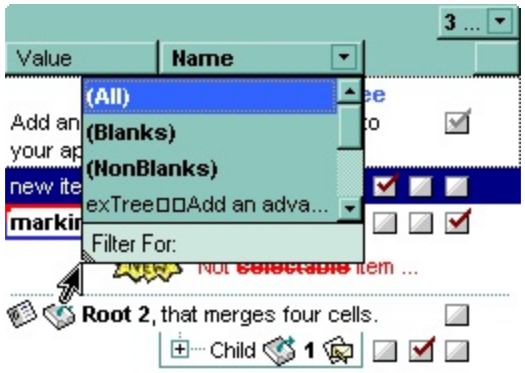
The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. The filter prompt feature allows you to filter the items as you type while the filter bar is visible on the bottom part of the list area. The Filter prompt feature allows at runtime filtering data on hidden columns too.

property Column.FilterBarDropDownWidth as Double

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

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

By default, the FilterBarDropDownWidth property is 1, and so, the width of the drop down filter window coincides with the width of the column. Use the Width property to specify the width of the column. Use FilterBarDropDownHeight property to specify the height of the drop down filter window. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the DisplayFilterButton property to display a filter button to the column's caption. Use the Description property to define predefined strings in the filter bar. At run-time, the user can resize the drop down filter window by clicking the left-bottom ticker. The FilterBarDropDownWidth property is changed if the user resizes the 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 Tree1.Columns(0)
   .FilterBarDropDownWidth = 2
End With
```

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

```
With Tree1.Columns(0)
    .FilterBarDropDownWidth = -150
End With
```

property Column.FilterList as FilterListEnum

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

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

By default, the FilterList property is exAllItems. Use the FilterList property to specify the items being included in the column's drop down filter list. Use the DisplayFilterButton property to display the column's filter bar button. The DisplayFilterDate property specifies whether the drop down filter window displays a date selector to specify the interval dates to filter for. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter.

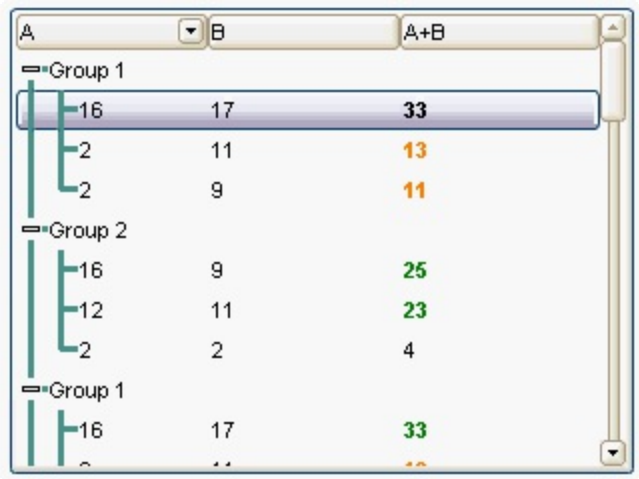
property Column.FilterOnType as Boolean

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

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

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

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



Steps:

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

property Column.FilterType as FilterTypeEnum

Specifies the column's filter type.

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

The FilterType property defines the filter's type. By default, the FilterType is exAll. No filter is applied if the FilterType is exAll. The Filter property defines the column's filter. Use the DisplayFilterButton property to display the column's filter button.

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 FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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.

property Column.FireFormatColumn as Boolean

Retrieves or sets a value that indicates whether the control fires FormatColumn to format the caption of a cell hosted by 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 in the list. For instance, the FormatColumn event may be useful for formatting the column of prices by grouping digits, and displaying the current currency. Newer versions of the component provides the FormatColumn property that helps formatting a cell using the several predefined functions without using the control's event FormatColumn.

property Column.FormatColumn as String

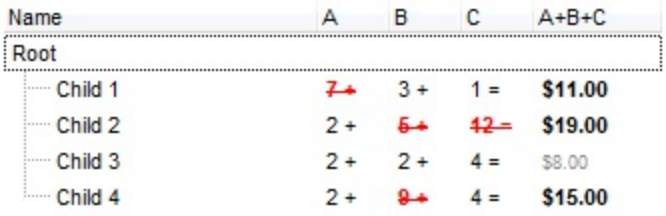
Specifies the format to display the cells in the column.

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><font ;7>' : '') + currency(=:0)*" displays the cells that contains not empty daya, the value in currency format, with a different font and color for values less than 10, and bolded for those that are greater than 10, as can see in the following screen shot in the column (A+B+C):



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

The expression supports cell's identifiers as follows:

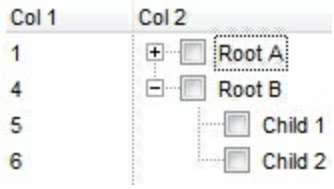
- ***%0**, **%1**, **%2**, ...{any} specifies the value of the cell in the column with the index 0, 1 2, ... The CellCaption property defines the cell's value. For example, "%0 format ``" formats the value in the cell at index 0 using the current regional settings, while "int(%1)" converts the value in the cell at index 1 to an integer.*
- ***%C0**, **%C1**, **%C2**, ...{string} specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property gets the cell's formatted caption. The cell's displayed string may differ from its actual value. For example, if a cell displays HTML content, %0 returns the HTML format including the tags, while %C0 returns the cell's content as a plain string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell at index 1 to uppercase, while "%C0 left 2" returns the leftmost two characters of the caption in the cell at index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ...{any} specifies the cell's extra data in the column with the index 0, 1 2, ... The CellData property associates any extra/user data to a cell. For example, "%CD0 = your user data" specifies all cells in the column with index 0 whose CellData property is equal to your user data.*
- ***%CS0**, **%CS1**, **%CS2**, ...{number} specifies the cell's state in the column with the index 0, 1 2, ... The CellState property defines the state of a cell, indicating whether it is checked or unchecked. For example, "%CS0" identifies all checked items in the column with index 0, while "not %CS1" identifies all unchecked items in the column with index 1.*
- ***%CC0**, **%CC1**, **%CC2**, ... {number} retrieve the number of child items (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ChildCount property returns the number of child items. For example, "%CC0" identifies all parent items, while "%CC0 = 0" identifies all leaf items.*
- ***%CX0**, **%CX1**, **%CX2**, ... {boolean} returns true if the item hosting the cell is expanded, or false if it is collapsed (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ExpandItem property specifically indicates whether the item is expanded or collapsed. For example, "%CX0" refers to all expanded items, while "not %CX0" identifies all collapsed items*

The predefined operators for auto-numbering are:

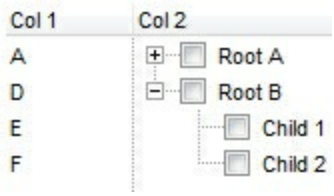
- number **index** 'format', indicates the index of the item. The first added item has the index 0, the second added item has the index 1, and so on. The index of the item remains the same even if the order of the items is changed by sorting. For instance, 1 index '' gets the index of the item starting from 1 while 100 index '' gets the index of the item starting from 100. The number indicates the starting index, while the format is a set of characters to be used for specifying the index. If the format is missing, the index of the item is formatted as numbers. For instance: 1 index 'A-Z' gets the index as A, B, C... Z, BA, BB, ... BZ, CA, ... The 1 index 'abc' gives the index as:

a,b,c,ba,bb,bc,ca,cb,cc,.... You can use other number formatting function to format the returned value. For instance "1 index " format '0||2|:'" gets the numbers grouped by 2 digits and separated by : character.

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

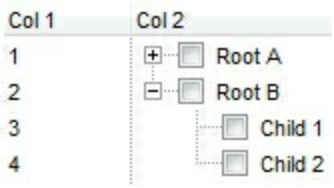


In the following screen shot the FormatColumn("Col 1") = "1 index 'A-Z'"

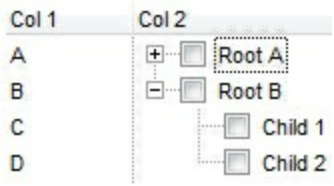


- number **apos** 'format' indicates the absolute position of the item. The first displayed item has the absolute position 0 (scrolling position on top), the next visible item is 1, and so on. The number indicates the starting position, while the format is a set of characters to be used for specifying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

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



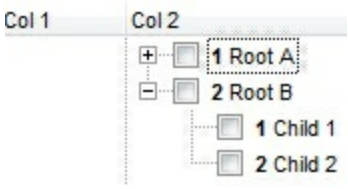
  In the following screen shot the FormatColumn("Col 1") = "1 apos 'A-Z'"



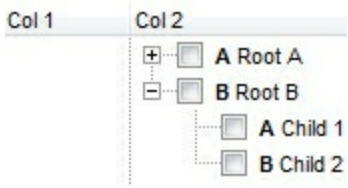
- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for

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

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

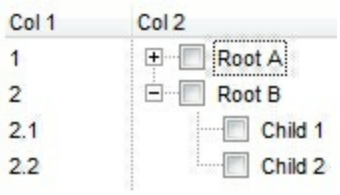


In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos 'A-Z' + ' ' + value"

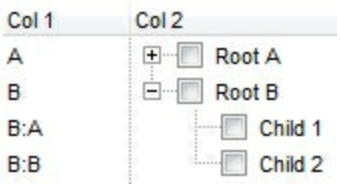


- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position.For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100. If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied, the opos gets the position of the item in the list of unfiltered items, while the pos gets the position of the item in the filtered list.*
- number **rpos** 'format' indicates the relative recursive position of the item. The recursive position indicates the position of the parent items too. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is of the following type "delimiter|format|format|...". If the format is missing, the delimiter is . character, and the positions are formatted as numbers. The format is applied consecutively to each parent item, from root to item itself.

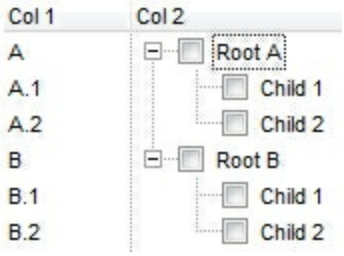
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ""



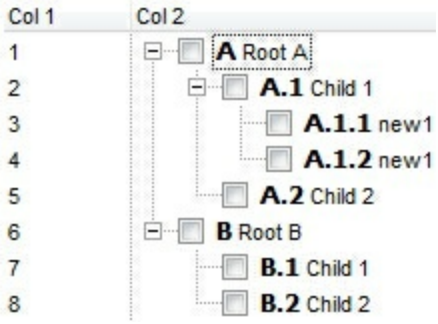
In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"



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



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



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

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

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

```
With Tree1
  .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 AxTree1
  .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 'EXTREELib' for the library: 'ExTree 1.0 Control Library'

  #import "C:\\Windows\\System32\\ExTree.dll"
  using namespace EXTREELib;
*/
EXTREELib::ITreePtr spTree1 = GetDlgItem(IDC_TREE1)->GetControlUnknown();
((EXTREELib::IColumnPtr)(spTree1->GetColumns()->Add(L"Currency")))-
>PutFormatColumn(L"currency(dbl(value))");
EXTREELib::IItemsPtr var_Items = spTree1->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:

```
(axTree1.Columns.Add("Currency") as EXTREELib.Column).FormatColumn =
"currency(dbl(value))";
EXTREELib.Items var_Items = axTree1.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.Tree1
    .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 specifies the alignment of the column's caption.

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

property Column.HeaderBold as Boolean

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

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

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

property Column.HeaderImage as Long

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

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

Use the HeaderImage property to assign an icon to the column's header. Use the HeaderImageAlignment property to align the column's icon inside the column's header.

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 in the column's header.

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

property Column.HeaderItalic as Boolean

Retrieves or sets the Italic 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 caption should appear in italic.

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

property Column.HeaderStrikeOut as Boolean

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

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

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

property Column.HeaderUnderline as Boolean

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

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

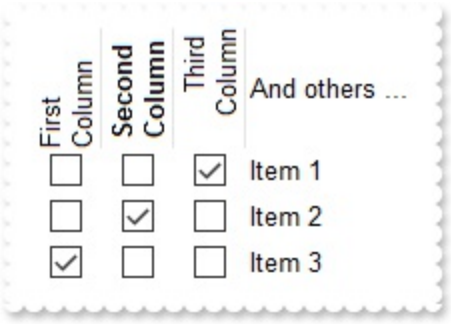
Use the HeaderUnderline property to specify whether the column's caption should appear in underline. Use the CellUnderline or ItemUnderline properties to specify whether the cell or the item should appear in underline. Use the HeaderItalic, HeaderBold or HeaderStrikeOut property to specify different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

property Column.HeaderVertical as Boolean

Specifies whether the column's header is vertically displayed.

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

Use the HeaderVertical property to display vertically the column's caption. Use the HeaderAlignment property to align the caption in the column's header. Use the Caption property to assign a caption to a column. Use the HTMLCaption property to specify an HTML caption to a column. Use the HeaderImage property to assign an icon to a column.



property Column.HTMLCaption as String

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

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

If the HTMLCaption property is empty, the Caption property is displayed in the column's header. If the HTMLCaption property is not empty, the control uses it when displaying the column's header. Use the HeaderHeight property to change the height of the control's header bar. The list of built-in HTML tags supported are here. Use the **<img>** HTML tag to insert icons or picture inside the column's caption.

property Column.Index as Long

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

Type	Description
Long	A long expression that represents the index of an object in a 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 the column's key.

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

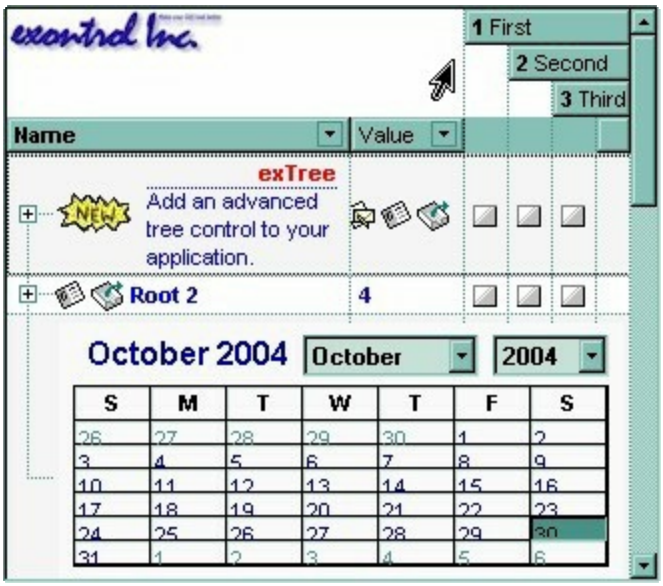
The column's key defines a column when using the Item property. Use the Index or the Key property to identify a column, when using the Columns property.

property Column.LevelKey as Variant

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

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

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



property Column.MaxWidthAutoResize as Long

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

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

Use the MaxWidthAutoResize property to set the maximum column's width while the WidthAutoResize property is True. If the MaxWidthAutoResize property is less than zero, there is no maximum value for the column's width. By default, the MaxWidthAutoResize property is -1. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area.

property Column.MinWidthAutoResize as Long

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

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

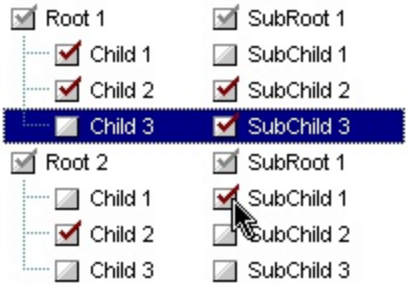
Use the MinWidthAutoResize property to set the minimum column's width while the WidthAutoResize property is True. Use the Width property to specify the column's width. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area.

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.

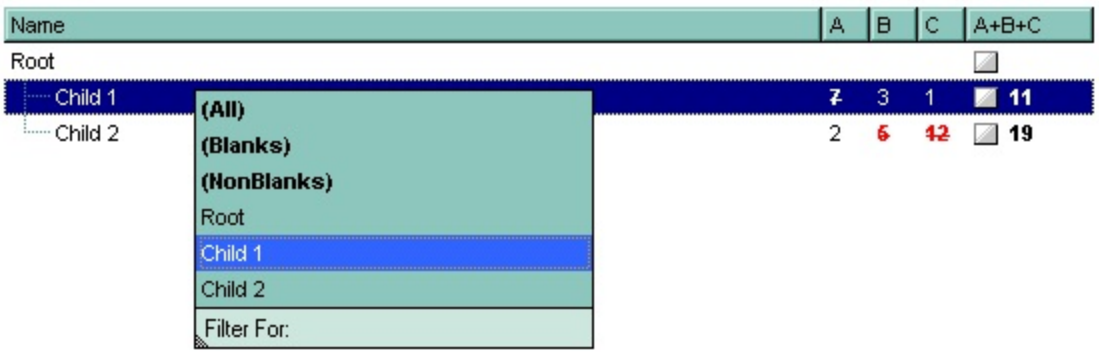
The column's index is not the same with the column's position. The Index property of Column cannot be changed by the user. Use the Position property to change the column's position. The EnsureVisibleColumn method ensures that a given column fits the control's client area. Use the SortPosition property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

method Column.ShowFilter ([Options as Variant])

Shows the column's filter window.

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

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



For instance, the following VB sample displays the column's drop down filter window when

the user right clicks the control:

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   If (Button = 2) Then
      With Tree1.Columns
         With .Item(Tree1.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 AxTree1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseUpEvent) Handles AxTree1.MouseUpEvent
   If (e.button = 2) Then
      With AxTree1.Columns
         With .Item(AxTree1.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 axTree1_MouseUpEvent(object sender,
AxEXTREELib._ITreeEvents_MouseUpEvent e)
{
   if (e.button == 2)
   {
      EXTREELib.Column c = axTree1.Columns[axTree1.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 OnMouseUpTree1(short Button, short Shift, long X, long Y)
{
    m_tree.GetColumns().GetItem( COleVariant( m_tree.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.Tree1.Columns
        With .Item(thisform.Tree1.ColumnFromPoint(-1, 0))
            .ShowFilter("-1,-1,200,200")
        EndWith
    EndWith
endif
```

property Column.SortOrder as SortOrderEnum

Specifies the column's sort order.

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

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

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::

  ```
  Tree1.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 sort descending the list of root items on the "Column 1"(if your control displays a list, all items are considered being root items).

  ```
  Tree1.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.

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.

The SortType property specifies how the column gets sorted. By default, the column's SortType is String. The CellCaption property indicates the values being sorted. 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. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. The SortOrder property determines the column's sort order. The Sort event is fired when the user sorts a column. The SortPosition property changes the position of the column in the sorting columns collection. The CellData property specifies the values being sorted, if the SortType property is SortUserData, SortUserDataString.

property Column.ToolTip as String

Specifies the column's tooltip description.

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

By default, the ToolTip property is "..." (three dots). Use the ToolTip property to assign a tooltip to a column. If the ToolTip property is "...", the control displays the column's caption if it doesn't fit the column's header. Use the Caption or HTMLCaption property to specify the caption of the column. The column's tooltip shows up when the cursor hovers the column's header. 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 resize the column. The ColumnAutoResize property specifies whether the visible columns fit the control's client area. Use the Position property to specify the column's position. Use the HeaderVisible property to show or hide the control's header bar. Use the ColumnFromPoint property to get the column from point. Use the Remove method to remove a column.

property Column.Width as Long

Retrieves or sets the column's width.

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

The Width property specifies the column's width in pixels. Use the Visible property to hide a column. Use the SortBarColumnWidth property to specify the column's head in the control's sort bar. Use the ColumnAutoResize property to fit all visible columns in the control's client area. Use the FilterBarDropDownWidth property to specify the width of the drop down filter window.

The following VB sample shows how to set the width for all columns:

```
Private Sub Tree1_AddColumn(ByVal Column As EXTREELibCtl.IColumn)
   Column.Width = 128
End Sub
```

The following VB.NET sample changes the column's width when a new column is added:

```
Private Sub AxTree1_AddColumn(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_AddColumnEvent) Handles AxTree1.AddColumn
   e.column.Width = 128
End Sub
```

The following C# sample changes the column's width when a new column is added:

```
private void axTree1_AddColumn(object sender,
AxEXTREELib._ITreeEvents_AddColumnEvent e)
{
   e.column.Width = 128;
}
```

The following C++ sample changes the column's width when a new column is added:

```
#include "Column.h"
#include "Columns.h"
void OnAddColumnTree1(LPDISPATCH Column)
{
```

```
    CColumn column( Column );
    column.SetWidth( 128 );
}
```

The following VFP sample changes the column's width when a new column is added:

```
*** ActiveX Control Event ***
LPARAMETERS column

with column
    .Width = 128
endwith
```

property Column.WidthAutoResize as Boolean

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

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

If the WidthAutoResize property is True, the column's width is resized after user expands, or collapse the items. Also, the column's width is refreshed if the user adds new items to the control. If the WidthAutoResize property is True, the column's width is not larger than MaxWidthAutoResize value, and it is not less than MinWidthAutoResize value. You can use the AutoWidth property to computes the column's width to fit its content. For instance, if you have a tree with one column, and this property True, you can simulate a simple tree, because the control will automatically add a horizontal scroll bar when required. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area.

Columns object

The ExTree control supports multiple 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's columns collection is empty

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 caption for the column being added

Return	Description
Variant	A Column object that indicates the newly added column.

By default, the control contains no columns. Before adding new items, you need to add columns. Use the Add property to add new columns to the control. The control fires the AddColumn event is fired when a new columns has been added to Columns collection. Use the Caption property to change the column's caption. Use the HTLMCaption property to display the column's caption using HTML tags. To hide a column use the Visible property of the Column object. Use the AddItem, InsertItem, InsertControlItem, 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 ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample adds columns from a record set:

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Orders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Opens the table using static mode
Tree1.BeginUpdate
' Add the columns
With Tree1.Columns
For Each f In rs.Fields
   .Add f.Name
Next
End With
Tree1.PutItems rs.getRows()
Tree1.EndUpdate
```

The following VC sample adds a column:

```
#include "Columns.h"
#include "Column.h"
```

```
CColumns columns = m_tree.GetColumns();
CColumn column( V_DISPATCH( &columns.Add( "Column 1" ) ) );
column.SetHeaderBold( TRUE );
```

The following VB.NET sample adds a column:

```
With AxTree1.Columns
   With .Add("Column 1")
      .HeaderBold = True
   End With
End With
```

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

```
With AxTree1.Columns
   Dim c As EXTREELib.Column
   c = .Add("Column 1")
   With c
      .HeaderBold = True
   End With
End With
```

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

The following C# sample adds a column:

```
EXTREELib.Column column = axTree1.Columns.Add( "Column 1" ) as EXTREELib.Column;
column.HeaderBold = true;
```

The following VFP sample adds a column:

```
with thisform.Tree1.Columns.Add( "Column 1" )
   .HeaderBold = .t.
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 to remove all columns in the control. The Clear method removes all items, too. Use the RemoveAllItems method to remove all items in the control.

property Columns.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	Counts the Column object into the collection.

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

The following VB sample enumerates the columns in the control:

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

The following VB sample enumerates the columns in the control:

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

The following VC sample enumerates the columns in the control:

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

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

```
With AxTree1.Columns
   Dim i As Integer
   For i = 0 To .Count - 1
```

```
    Debug.WriteLine(.Item(i).Caption)
  Next
End With
```

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

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

The following VFP sample enumerates the columns in the control:

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

property Columns.Item (Index as Variant) as Column

Returns a specific Column of the Columns collection.

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

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 equivalents:

```
Tree1.Columns.Item ("Freight")
Tree1.Columns ("Freight")
```

The following VB sample enumerates the columns in the control:

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

The following VC sample enumerates the columns in the control:

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

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

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

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

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

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

The following VFP sample enumerates the columns in the control:

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

property Columns.ItemBySortPosition (Position as Variant) as Column

Returns a Column object giving its sorting position.

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

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

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

```
Dim s As String, i As Long, c As Column
i = 0
With Tree1.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_tree.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 AxTree1
    Dim s As String, i As Integer, c As EXTREELib.Column
    i = 0
    With AxTree1.Columns
        c = .ItemBySortPosition(i)
        While (Not c Is Nothing)
            s = s + """" & c.Caption & """ " & IIf(c.SortOrder =
EXTREELib.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;
EXTREELib.Column column = axTree1.Columns.get_ItemBySortPosition( i );
while ( column != null )
{
    strOutput += column.Caption + " " + ( column.SortOrder ==
EXTREELib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
    column = axTree1.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.Tree1.Columns
   c = .ItemBySortPosition(i)
   do While (!isnull(c))
      with c
         s = s + "'" + .Caption
         s = s + "' " + Iif(.SortOrder = 1, "A", "D") + " "
         i = i + 1
      endwith
      c = .ItemBySortPosition(i)
   enddo
endwith
s = "Sort: " + s
wait window nowait s
```

method Columns.Remove (Index as Variant)

Removes a specific member from the Columns collection.

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

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

ConditionalFormat object

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

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

property ConditionalFormat.ApplyTo as FormatApplyToEnum

Specifies whether the format is applied to items or columns.

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

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

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

```
With Tree1.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_tree.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 AxTree1.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):

```
EXTREELib.ConditionalFormat cf = axTree1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXTREELib.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.Tree1.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 Tree1.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_tree.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 AxTree1.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):

```
EXTREELib.ConditionalFormat cf = axTree1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXTREELib.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.Tree1.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). Use the Valid property checks 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 may be a combination of variables, constants, strings, dates and operators. 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 two # characters, ie #1/31/2001 10:00# means the January 31th, 2001, 10:00 AM.

The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ...{any} specifies the value of the cell in the column with the index 0, 1 2, ... The CellCaption property defines the cell's value. For example, "%0 format ``" formats the value in the cell at index 0 using the current regional settings, while "int(%1)" converts the value in the cell at index 1 to an integer.*
- ***%C0**, **%C1**, **%C2**, ...{string} specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property gets the cell's formatted caption. The cell's displayed string may differ from its actual value. For example, if a cell displays HTML content, %0 returns the HTML format including the tags, while %C0 returns the cell's content as a plain string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell at index 1 to uppercase, while "%C0 left 2" returns the leftmost two characters of the caption in the cell at index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ...{any} specifies the cell's extra data in the column with the*

index 0, 1 2, ... The CellData property associates any extra/user data to a cell. For example, "%CD0 = your user data" specifies all cells in the column with index 0 whose CellData property is equal to your user data.

- ***%CS0**, **%CS1**, **%CS2**, ...{number} specifies the cell's state in the column with the index 0, 1 2, ... The CellState property defines the state of a cell, indicating whether it is checked or unchecked. For example, "%CS0" identifies all checked items in the column with index 0, while "not %CS1" identifies all unchecked items in the column with index 1.*
- ***%CC0**, **%CC1**, **%CC2**, ... {number} retrieve the number of child items (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ChildCount property returns the number of child items. For example, "%CC0" identifies all parent items, while "%CC0 = 0" identifies all leaf items.*
- ***%CX0**, **%CX1**, **%CX2**, ... {boolean} returns true if the item hosting the cell is expanded, or false if it is collapsed (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ExpandItem property specifically indicates whether the item is expanded or collapsed. For example, "%CX0" refers to all expanded items, while "not %CX0" identifies all collapsed items*

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

Samples:

1. "**1**", highlights all cells or items. Use this form, when you need to highlight all cells or items in the column or control.
2. "**%0 >= 0**", highlights the cells or items, when the cells in the first column have the value greater or equal with zero
3. "**%0 = 1 and %1 = 0**", highlights the cells or items, when the cells in the first column have the value equal with 0, and the cells in the second column have the value equal with 0
4. "**%0+%1>%2**", highlights the cells or the items, when the sum between first two columns is greater than the value in the third column
5. "**%0+%1 > %2+%3**", highlights the cells or items, when the sum between first two columns is greater than the sum between third and forth column.
6. "**%0+%1 >= 0 and (%2+%3)/2 < %4-5**", highlights the cells or the items, when the sum between first two columns is greater than 0 and the half of the sum between third and forth columns is less than fifth column minus 5.
7. "**%0 startwith 'A'**" specifies the cells that starts with A
8. "**%0 endwith 'Bc'**" specifies the cells that ends with Bc
9. "**%0 contains 'aBc'**" specifies the cells that contains the aBc string
10. "**lower(%0) contains 'abc'**" specifies the cells that contains the abc, AbC, ABC, and so on

11. "**upper(%0)**" retrieves the uppercase string
12. "**len(%0)>0**" specifies the not blanks cells
13. "**len %0 = 0**" specifies the blanks cells

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

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

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

```
Tree1.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_tree.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:

```
AxTree1.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:

```
axTree1.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.Tree1.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 Tree1.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 Tree1.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_tree.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 AxTree1.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):

```
EXTREELib.ConditionalFormat cf = axTree1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Italic = true;
cf.ApplyTo = (EXTREELib.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.Tree1.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 Tree1.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_tree.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 AxTree1.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):

```
EXTREELib.ConditionalFormat cf = axTree1.ConditionalFormats.Add("%1+%2<%0",null);
```

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

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

```
with thisform.Tree1.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 Tree1.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_tree.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 AxTree1.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):

```
EXTREELib.ConditionalFormat cf = axTree1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Underline = true;
cf.ApplyTo = (EXTREELib.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.Tree1.ConditionalFormats.Add("%1+%2<%0")
   .Underline = .t.
   .ApplyTo = 1
endwith
```

property ConditionalFormat.Valid as Boolean

Checks whether the expression is syntactically correct.

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

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

ConditionalFormats object

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

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

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

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

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

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

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

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

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

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

```
Tree1.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 Tree1.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_tree.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_tree.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:

```
AxTree1.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 AxTree1.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:

```
axTree1.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):

```
EXTREELib.ConditionalFormat cf = axTree1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXTREELib.FormatApplyToEnum)1;
```

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

```
thisform.Tree1.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.Tree1.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 Tree1.ConditionalFormats
   Debug.Print c.Expression
Next
```

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

```
Dim i As Integer
With Tree1.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_tree.GetConditionalFormats().GetCount(); i++ )
{
   CConditionalFormat cf = m_tree.GetConditionalFormats().GetItem( COleVariant( i ) );
   OutputDebugString( cf.GetExpression() );
}
```

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

```
Dim c As EXTREELib.ConditionalFormat
For Each c In AxTree1.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 AxTree1.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 (EXTREELib.ConditionalFormat c in axTree1.ConditionalFormats)
   System.Diagnostics.Debug.Write(c.Expression);
```

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

```
for (int i = 0; i < axTree1.ConditionalFormats.Count; i++)
   System.Diagnostics.Debug.Write(axTree1.ConditionalFormats[i].Expression);
```

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

```
with thisform.Tree1.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 Tree1.ConditionalFormats
   Debug.Print c.Expression
Next
```

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

```
Dim i As Integer
With Tree1.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_tree.GetConditionalFormats().GetCount(); i++ )
{
   CConditionalFormat cf = m_tree.GetConditionalFormats().GetItem( COleVariant( i ) );
   OutputDebugString( cf.GetExpression() );
}
```

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

```
Dim c As EXTREELib.ConditionalFormat
```

```
For Each c In AxTree1.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 AxTree1.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 (EXTREELib.ConditionalFormat c in axTree1.ConditionalFormats)
   System.Diagnostics.Debug.Write(c.Expression);
```

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

```
for (int i = 0; i < axTree1.ConditionalFormats.Count; i++)
   System.Diagnostics.Debug.Write(axTree1.ConditionalFormats[i].Expression);
```

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

```
with thisform.Tree1.ConditionalFormats
   for i = 0 to .Count - 1
      wait .Item(i).Expression
   next
endwith
```

method ConditionalFormats.Remove (Key as Variant)

Removes a specific member from the collection.

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

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

ExDataObject object

The OleDragDrop event notifies your application that the user drags some data on the control. Defines the object that contains OLE drag and drop information. The ExDataObject object supports the following method and properties:

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

method ExDataObject.Clear ()

Deletes the contents of the DataObject object.

Type	Description

The Clear method can be called only for drag sources. The OleDragDrop event notifies your application that the user drags some data on the control.

property ExDataObject.Files as ExDataObjectFiles

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

Type	Description
ExDataObjectFiles	An ExDataObjectFiles object that contains a list of filenames used in OLE drag and drop operations.

The Files property is valid only if the format of the clipboard data is exCFFiles. The OleDragDrop event notifies your application that the user drags some data on the control.

method ExDataObject.GetData (Format as Integer)

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

Type	Description
Format as Integer	An exClipboardFormatEnum expression that defines the data's format

Return	Description
Variant	A Variant value that contains the ExDataObject's data in the given format

Use GetData property to retrieve the clipboard's data that has been dragged to the control. It's possible for the GetData and SetData methods to use data formats other than exClipboardFormatEnum , including user-defined formats registered with Windows via the RegisterClipboardFormat() API function. The GetData method always returns data in a byte array when it is in a format that it is not recognized. Use the Files property to retrieves the filenames if the format of data is exCFFiles

method ExDataObject.GetFormat (Format as Integer)

Returns a value indicating whether the ExDataObject's data is of specified format.

Type	Description
Format as Integer	A constant or value that specifies a clipboard data format like described in exClipboardFormatEnum enum.

Return	Description
Boolean	A boolean value that indicates whether the ExDataObject's data is of specified format.

Use the GetFormat property to verify if the ExDataObject's data is of a specified clipboard format. The GetFormat property retrieves True, if the ExDataObject's data format matches the given data format.

method ExDataObject.SetData ([Value as Variant], [Format as Variant])

Inserts data into a ExDataObject object using the specified data format.

Type	Description
Value as Variant	A data that is going to be inserted to ExDataObject object.
Format as Variant	A constant or value that specifies the data format, as described in exClipboardFormatEnum enum

Use SetData property to insert data for OLE drag and drop operations. Use the Files property is you are going to add new files to the clipboard data. The OleDragDrop event notifies your application that the user drags some data on the control.

ExDataObjectFiles object

The ExDataObjectFiles contains a collection of filenames. The ExDataObjectFiles object is used in OLE Drag and drop events. In order to get the list of files used in drag and drop operations you have to use the Files property. The OleDragDrop event notifies your application that the user drags some data on the control. The ExDataObjectFiles object supports the following properties and methods:

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

method ExDataObjectFiles.Add (FileName as String)

Adds a filename to the Files collection

Type	Description
FileName as String	A string expression that indicates a filename.

Use Add method to add your files to ExDataObject object. The OleStartDrag event notifies your application that the user starts dragging items.

method ExDataObjectFiles.Clear ()

Removes all file names in the collection.

Type	Description

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

property ExDataObjectFiles.Count as Long

Returns the number of file names in the collection.

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

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

property ExDataObjectFiles.Item (Index as Long) as String

Returns a specific file name given its index.

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

method ExDataObjectFiles.Remove (Index as Long)

Removes a specific file name given its index into collection.

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

Use Clear method to remove all filenames,.

Items object

The Items object contains a collection of items. Each item is identified by 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 in the control. To access the Items collection use Items property of the control. Using the Items collection you can add, remove or change the control items. The Items collection can be organized as a hierarchy or as a tabular data. The Items collection 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 indicates whether the cell's caption should appear in bold.
CellButtonAutoWidth	Retrieves or sets a value indicating whether the cell's button fits the cell's caption.
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 the extra data for a specific cell.
CellEnabled	Returns or sets a value that determines whether a cell can respond to user-generated events.
CellFont	Retrieves or sets the cell's font.
CellForeColor	Retrieves or sets the cell's foreground color.
CellHAlignment	Retrieves or sets a value that indicates the alignment of the cell's caption.
CellHasButton	Retrieves or sets a value indicating whether the cell has associated a push button or not.
CellHasCheckBox	Retrieves or sets a value indicating whether the cell has associated a checkbox or not.
CellHasRadioButton	Retrieves or sets a value indicating whether the cell has associated a radio button or not.
CellHyperLink	Specifies whether the cell's is highlighted when the cursor

	mouse is over the cell.
CellImage	Retrieves or sets an Image that is displayed on the cell's area.
CellImages	Specifies an additional list of icons shown in the cell.
CellItalic	Retrieves or sets a value that indicates whether the cell's caption should appear in italic.
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.
CellParent	Retrieves the parent of an inner cell.
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 check and radio cells.
CellStrikeOut	Retrieves or sets a value that indicates whether the cell's caption should appear in strikeout.
CellToolTip	Retrieves or sets a text that is used to show the tooltip's cell.
CellUnderline	Retrieves or sets a value that indicates whether the cell's caption should appear in underline.
CellVAlignment	Retrieves or sets a value that indicates how the cell's caption is vertically aligned.
CellWidth	Retrieves or sets a value that indicates the width of the inner cell.
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.
Edit	Edits a cell.
EnableItem	Returns or sets a value that determines whether a item 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 colum. The searching starts at StartIndex item.
FindItemData	Finds the item giving its data.
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.
InnerCell	Retrieves the inner cell.
InsertControlItem	Inserts a new item of ActiveX type, and returns a handle to the newly created item.
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.
ItemAppearance	Specifies the item's appearance when the item hosts an ActiveX control.
ItemBackColor	Retrieves or sets a background color for a specific item.

ItemBold	Retrieves or sets a value that indicates whether the item should appear in bold.
ItemByIndex	Retrieves the handle of the item given its index in Items collection..
ItemCell	Retrieves the cell's handle based on a specific column.
ItemChild	Retrieves the child of a specified item.
ItemControlID	Retrieves the item's control identifier that was used by InsertControlItem.
ItemCount	Retrieves the number of items.
ItemData	Retrieves or sets the 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 indicates whether the item should appear in italic.
ItemMaxHeight	Retrieves or sets a value that indicates the maximum height when the item's height is variable.
ItemMinHeight	Retrieves or sets a value that indicates the minimum height when the item's height is sizing.
ItemObject	Retrieves the ActiveX object associated, if the item was created using InsertControlItem method.
ItemParent	Returns the handle of 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 indicates 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 indicates whether the item should appear in underline.
ItemWidth	Retrieves or sets a value that indicates the item's width while it contains an ActiveX control.
ItemWindowHost	Retrieves the window's handle that hosts an ActiveX control when the item was created using InsertControlItem.
ItemWindowHostCreateStyle	Retrieves or sets a value that indicates a combination of window styles used to create the ActiveX window host.
LastVisibleItem	Retrieves the handle of the last visible item.
LockedItem	Retrieves the handle of the locked/fixed 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 handle of selected item giving its index in selected items collection.
SelectedItem	Retrieves the selected item's handle given its index in 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.
SplitCell	Splits a cell, and returns the inner created cell.
UnmergeCells	Unmerges a list of cells.
UnselectAll	Unselects all items.
UnsplitCell	Unsplits a cell.
VisibleCount	Retrieves the number of visible items.
VisibleItemCount	Retrieves the number of visible items.

property Items.AcceptSetParent (Item as HITEM, NewParent as HITEM) as Boolean

Retrieves a value indicating whether the SetParent method can be accomplished.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being moved.
NewParent as HITEM	A long expression that indicates the handle of the parent item where the item should be moved.
Boolean	A boolean expression that indicates whether the item can be child of the NewParent item.

Use this property to make sure that SetParent can be called. The AcceptSetParent property checks if an item can be child of another item.

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 cell's caption for the first column. or a safe array that contains the captions for each column. The Caption accepts HTML format, if the CellCaptionFormat property is exHTML.

Return	Description
HITEM	A long expression that indicates the handle of the newly created item.

Use the Add method to add new columns to the control. If the control contains no columns, the AddItem method fails. Use the AddItem property when your control acts like a list. Use InsertItem when your control acts like a tree. Use the InsertControlItem property when the item needs to host an ActiveX control. Use the LockedItemCount property to add or remove items locked to the top or bottom side of the control. Use the MergeCells method to combine two or multiple cells in a single cell. Use the SplitCell property to split a cell. Use the BeginUpdate and EndUpdate methods to maintain performance while adding new columns and items. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula. Use the FormatColumn property to format the column.

The AddItem property adds a new item that has no parent. When a new item is added (inserted) to the Items collection, the control fires the AddItem event. If the control contains more than one column use the CellCaption property to set the cell's caption. If there are no columns AddItem method fails.

The following VB sample uses the VB Array function to add two items:

```
With Tree1
  .BeginUpdate

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

    With .Items
      .AddItem Array("Item 1.1", "Item 1.2", "Item 1.3")
      .AddItem Array("Item 2.1", "Item 2.2", "Item 2.3")
```

```
    End With

  .EndUpdate
End With
```

In VB/NET using the /NET assembly, the Array equivalent is New Object such as follows:

```
With Tree1
  .BeginUpdate()

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

  With .Items
    .AddItem(New Object() {"Item 1.1", "Item 1.2", "Item 1.3"})
    .AddItem(New Object() {"Item 2.1", "Item 2.2", "Item 2.3"})
  End With

  .EndUpdate()
End With
```

In C# using the /NET assembly, the Array equivalent is new object such as follows:

```
extree1.BeginUpdate();

extree1.Columns.Add("Column 1");
extree1.Columns.Add("Column 2");
extree1.Columns.Add("Column 3");

extree1.Items.AddItem(new object[] { "Item 1.1", "Item 1.2", "Item 1.3" });
extree1.Items.AddItem(new object[] { "Item 2.1", "Item 2.2", "Item 2.3" });

extree1.EndUpdate();
```

Use the PutItems method to load an array, like in the following VB sample:

```
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
Tree1.BeginUpdate
' Add the columns
With Tree1.Columns
For Each f In rs.Fields
   .Add f.Name
Next
End With
Tree1.PutItems rs.getRows()
Tree1.EndUpdate
```

The following C++ sample adds new items to the control:

```
#include "Items.h"
CItems items = m_tree.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 AxTree1.Items
   Dim iNewItem As Integer
   iNewItem = .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:

```
EXTREELib.Items items = axTree1.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:

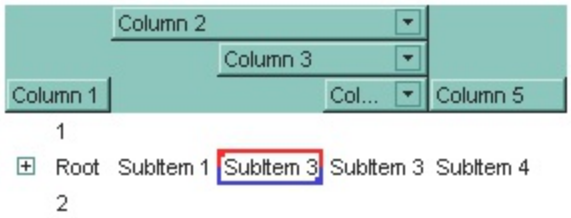
```
with thisform.Tree1.Items
   .DefaultItem = .AddItem("Item 1")
   .CellCaption(0, 1) = "SubItem 1"
endwith
```

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 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. The HTML colors are not applied if the item is selected. Use the SelectedItem property to specify whether an item is selected or unselected. Use the Def(exCellBackColor) property to specify the background color for all cells in the column. Use the Add method to add new skins to the control. You can define new skins and to use it to mark some cells, like in the following samples. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.



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

```
With Tree1
  With .VisualAppearance
    .Add &H40, App.Path + "\cell.ebn"
```

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

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

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

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

```
With AxTree1
    With .VisualAppearance
        .Add(&H40, "D:\Temp\ExTree.Help\cell.ebn")
    End With
    With .Items
        .CellBackColor(.FirstVisibleItem, 0) = &H40000000
    End With
End With
```

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

```
axTree1.VisualAppearance.Add(0x40, "D:\\Temp\\ExTree.Help\\cell.ebn");
axTree1.Items.set_CellBackColor(axTree1.Items.FirstVisibleItem, 0, 0x40000000);
```

The following VFP sample changes the cell's appearance.

```
With thisform.Tree1
    With .VisualAppearance
        .Add(64, "D:\Temp\ExTree.Help\cell.ebn")
    EndWith
    with .Items
        .DefaultItem = .FirstVisibleItem
```

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

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

```
axTree1.Items.set_CellBackColor(axTree1.Items.FocusItem, 0, ToUInt32(Color.Red));
```

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 VB.NET sample changes the background color for the focused cell:

```
With AxTree1.Items
  .CellBackColor(.FocusItem, 0) = ToUInt32(Color.Red)
End With
```

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 C++ sample changes the background color for the focused cell:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetCellBackColor( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),
```

```
RGB(255,0,0) );
```

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

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellBackColor( 0, 0 ) = RGB(255,0,0)
endwith
```

For instance, the following VB code changes background color of the left top cell of your control: Tree1.Items.CellBackColor(Tree.Items(0), 0) = vbBlue

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell (see ItemCell property). Here's few hints how to use properties with Item and ColIndex parameters:

Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True

Tree1.Items.CellBold(Tree1.Items(0), 0) = True

Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True

property Items.CellBold([Item as Variant], [ColIndex as Variant]) as Boolean

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

The following VB sample bolds the cells in the first column

```
Dim h As Variant
Tree1.BeginUpdate
With Tree1.Items
For Each h In Tree1.Items
   .CellBold(h, 0) = True
Next
End With
Tree1.EndUpdate
```

The following C++ sample bolds the focused cell:

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

The following C# sample bolds the focused cell:

```
axTree1.Items.set_CellBold(axTree1.Items.FocusItem, 0, true);
```

The following VB.NET sample bolds the focused cell:

```
With AxTree1.Items
   .CellBold(.FocusItem, 0) = True
End With
```

The following VFP sample bolds the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellBold( 0, 0 ) = .t.
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell (see ItemCell property). Here's few hints how to use properties with Item and ColIndex parameters:

Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True

Tree1.Items.CellBold(Tree1.Items(0), 0) = True

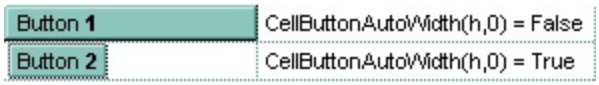
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True

property Items.CellButtonAutoWidth([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value indicating whether the cell's button fits the cell's caption.

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.
Boolean	A boolean expression indicating whether the cell's button fits the cell's caption.

By default, the CellButtonAutoWidth property is False. The CellButtonAutoWidth property has effect only if the CellHasButton property is true. Use the Def property to specify that all buttons in the column fit to the cell's content. If the CellButtonAutoWidth property is False, the width of the button is the same as the width of the column. If the CellButtonAutoWidth property is True, the button area covers only the cell's caption. Use the CellCaption property to specify the button's caption. Use the CellCaptionFormat property to assign an HTML caption to the button. The control fires the CellButtonClick property when the user clicks a button.



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 column's index, or the handle to the cell, if the Item parameter is 0, a string expression that indicates the column's caption or the column's key.
Variant	A variant expression that indicates the cell's caption. The cell's caption supports built-in HTML format.

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

Let's assume the following template in order to run the samples that follows:

```
BeginUpdate
LinesAtRoot = -1
FullRowSelect = False
Images("gBJJgBAIDAALAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaGEaAIAEEbjMjlErlktl(

SelBackColor = RGB(40,150,255)
BackColor = RGB(255,255,255)
```

```
Columns
{
   "Column 1"
   {
      Def(0) = True
   }
}
Items
{
   Dim h, h1
   h = AddItem("Counter132")
   CellCaptionFormat(h,0) = 1
   h1 = InsertItem(h,,"Right321")
   CellCaptionFormat(h1,0) = 1
   h1 = InsertItem(h,,"3Left, the next part should break the line
21 second line")
   CellSingleLine(h1,0) = False
   CellCaptionFormat(h1,0) = 1
   h1 = InsertItem(h,,"321Left")
   CellCaptionFormat(h1,0) = 1
   ExpandItem(h) = True
}
EndUpdate
```

The following samples uses the ItemFromPoint method to determine the hit test code from the point. The exHTCellCaptionIcon indicates that the cursor hovers an icon inside the cell's caption.

The following VB sample displays the index of the icon being clicked when the cell's caption includes <img> tags:

```
Private Sub Tree1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim i As HITEM, h As HitTestInfoEnum, c As Long
   With Tree1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
   End With
   If (i <> 0) Then
      If exHTCellCaptionIcon = (h And exHTCellCaptionIcon) Then
```

```
        Debug.Print "The index of icon being clicked is: " & (h And &HFFFF0000) / 65536 &
" Hex: " & Hex(h)
      End If
   End If
End Sub
```

The following VB sample changes the icon being clicked, by replacing the <img> tag in the CellCaption property (if your application have access to some regular expression or an easiest way to replace strings, the idea is to replace the n-th <img>..</img> tag element with a new value, and so the icon is changed):

```
Private Sub Tree1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim i As HITEM, h As HitTestInfoEnum, c As Long
   With Tree1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
      If (i <> 0) Then
         If exHTCellCaptionIcon = (h And exHTCellCaptionIcon) Then
            Dim iImg As Long
            iImg = (h And &HFFFF0000) / 65536
            With .Items
               Dim cellStr As String
               cellStr = .CellCaption(i, c)

               ' Replaces the iImg-th <img> tag in the CellCaption with a new value
               Dim n() As String
               n = Split(cellStr, "<img>")
               cellStr = ""
               For j = LBound(n) To UBound(n)
                  Dim sNext As String
                  sNext = n(j)
                  If (j > LBound(n)) Then
                     cellStr = cellStr + "<img>"
                     If (j = iImg + 1) Then
                        Dim lIndex As Long, p As Long
                        p = InStr(1, sNext, "</img>")
                        lIndex = Left(sNext, p - 1)
                        lIndex = lIndex Mod 3 + 1
                        sNext = RTrim(LTrim(Str(lIndex))) + Mid(sNext, p)
```

```
                End If
              End If
              cellStr = cellStr + sNext
            Next

            .CellCaption(i, c) = cellStr
          End With
        End If
      End If
    End With
End Sub
```

The following C++ sample displays the index of the icon being clicked when the cell's caption includes <img> tags:

```
void OnMouseDownTree1(short Button, short Shift, long X, long Y)
{
  long c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
  if ( hItem != 0 )
    if ( /*exHTCellCaptionIcon*/0x414 == ( hit & /*exHTCellCaptionIcon*/0x414 ) )
    {
      CString strOutput;
      strOutput.Format( "The index of icon being clicked is: %i, Hit = %08X\n", hit >> 16,
hit );
      OutputDebugString( strOutput );
    }
}
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell (see ItemCell property). Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

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 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 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. The HTML colors are not applied if the item is selected. Use the SelectedItem property to specify whether an item is selected or unselected.

property Items.CellChecked (RadioGroup as Long) as HCELL

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

Type	Description
RadioGroup as Long	A long expression that indicates the radio group identifier.
HCELL	A long expression that identifies the handle of the cell that's checked in the specified radio group. To retrieve the handle of the owner item you have to use CellItem property.

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 Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.Items.CellHasRadioButton(Item, 0) = True
   Tree1.Items.CellRadioGroup(Item, 0) = 1234 ' The 1234 is arbirary and it represents the identifier for the radio group
End Sub

Private Sub Tree1_CellStateChanged(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
   Debug.Print "In the 1234 radio group the """ & Tree1.Items.CellCaption(, Tree1.Items.CellChecked(1234)) & """ is checked."
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_tree.GetItems();
m_tree.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_tree.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 OnCellStateChangedTree1(long Item, long ColIndex)
{
   CItems items = m_tree.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 AxTree1
   .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 AxTree1_CellStateChanged(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_CellStateChangedEvent) Handles AxTree1.CellStateChanged
    With AxTree1.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:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
for (int i = 0; i < items.ItemCount; i++)
{
    items.set_CellHasRadioButton(items[i], 0, true);
    items.set_CellRadioGroup(items[i], 0, 1234);
}
axTree1.EndUpdate();
```

```
private void axTree1_CellStateChanged(object sender,
AxEXTREELib._ITreeEvents_CellStateChangedEvent e)
{
    string strOutput = axTree1.Items.get_CellCaption( 0,
axTree1.Items.get_CellChecked(1234) ).ToString();
    strOutput += " state = " + axTree1.Items.get_CellState(e.item, e.colIndex).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.Tree1.BeginUpdate()
```

```
with thisform.Tree1.Items
   local i
   for i = 0 to .ItemCount - 1
      .DefaultItem = .ItemByIndex(i)
      .CellHasRadioButton( 0,0 ) = .t.
      .CellRadioGroup(0,0) = 1234
   next
endwith
thisform.Tree1.EndUpdate()
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell (see ItemCell property). Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellData([Item as Variant], [ColIndex as Variant]) as Variant

Retrieves or sets the extra data for 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 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. Use the SortUserData or SortUserDataString type to sort the column based on the CellData value. Use the CellCaption property to specify the cell's caption.

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell (see ItemCell property). Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

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 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: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell (see ItemCell property). Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

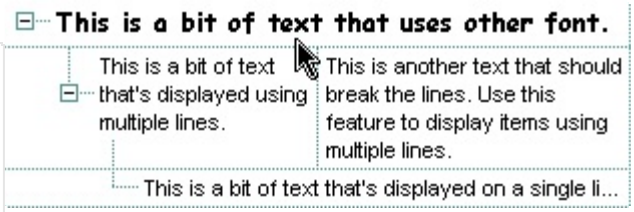
property Items.CellFont ([Item as Variant], [ColIndex as Variant]) as IFontDisp

Retrieves or sets the cell's font.

Type	Description
Item as Variant	A long expression that indicates the item's handle, or optional if the cell's handle is passed to ColIndex parameter
ColIndex as Variant	A long expression that indicates the column's index or cell's handle, or a string expression that indicates the column's caption.
IFontDisp	A Font object that indicates the cell's font.

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




```
With Tree1.Items
   .CellFont(.FocusItem, 0) = Tree1.Font
   With .CellFont(.FocusItem, 0)
      .Name = "Comic Sans MS"
      .Size = 10
      .Bold = True
   End With
End With
Tree1.Refresh
```

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

```
#include "Items.h"
#include "Font.h"
```

```
CItems items = m_tree.GetItems();
COleVariant vtItem(items.GetFocusItem()), vtColumn( (long)0 );
items.SetCellFont( vtItem, vtColumn, m_tree.GetFont().m_lpDispatch );
COleFont font = items.GetCellFont( vtItem, vtColumn );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_tree.Refresh();
```

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

```
With AxTree1.Items
   .CellFont(.FocusItem, 0) = IFDH.GetIFontDisp(AxTree1.Font)
   With .CellFont(.FocusItem, 0)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
AxTree1.CtlRefresh()
```

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:

```
axTree1.Items.set_CellFont( axTree1.Items.FocusItem, 0, IFDH.GetIFontDisp( axTree1.Font ) );
stdole.IFontDisp spFont = axTree1.Items.get_CellFont(axTree1.Items.FocusItem, 0 );
spFont.Name = "Comic Sans MS";
```

```
spFont.Bold = true;
axTree1.CtlRefresh();
```

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:

```
with thisform.Tree1.Items
    .DefaultItem = .FocusItem
    .CellFont(0,0) = thisform.Tree1.Font
    with .CellFont(0,0)
        .Name = "Comic Sans MS"
        .Bold = .t.
    endwith
endwith
thisform.Tree1.Object.Refresh()
```

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 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. The HTML colors are not applied if the item is selected. Use the SelectedItem property to specify whether an item is selected or unselected. 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:
Tree1.Items.CellForeColor(Tree1.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:

```
axTree1.Items.set_CellForeColor(axTree1.Items.FocusItem, 0, ToUInt32(Color.Red) );
```

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

```
With AxTree1.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_tree.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.Tree1.Items
   .DefaultItem = .FocusItem
   .CellForeColor( 0, 0 ) = RGB(255,0,0)
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellHAlignment ([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 CellHAlignment property aligns a particular cell. Use the Alignment property of the Column object to align all the cells in the column. Use the CellVAlignment property to align vertically the caption of the cell, when the item displays its content using multiple lines. Use the ClearCellHAlignment method to clear the cell's alignment previously set by the CellHAlignment property. If the CellHAlignment property is not set, the Alignment property of the Column object indicates the cell's alignment. If the cell belongs to the column that displays the hierarchy (TreeColumnIndex property), the cell can be aligned to the left or to the right.

The following VB sample right aligns the focused cell:

```
With Tree1.Items
   .CellHAlignment(.FocusItem, 0) = AlignmentEnum.RightAlignment
End With
```

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

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

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

```
With AxTree1.Items
   .CellHAlignment(.FocusItem, 0) = EXTREELib.AlignmentEnum.RightAlignment
End With
```

The following C# sample right aligns the focused cell:

```
axTree1.Items.set_CellHAlignment(axTree1.Items.FocusItem, 0,
EXTREELib.AlignmentEnum.RightAlignment);
```

The following VFP sample right aligns the focused cell:

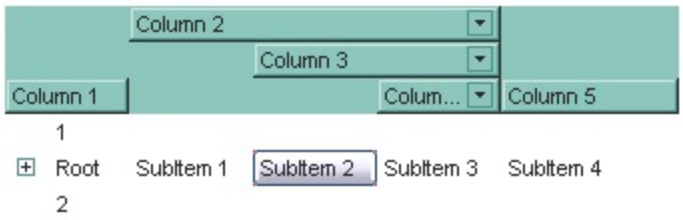
```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellHAlignment(0,0) = 2 && RightAlignment
endwith
```

property Items.CellHasButton([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value indicating whether the cell has associated a push button or not.

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.
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. See also: CellButtonAutoWidth property. Use the Add method to add new skins to the control. Use the Background property to specify a background color or a visual appearance for specific parts in the control.



The following VB sample changes the appearance for buttons in the cells. The sample use the skin "" when the button is up, and the skin "" when the button is down:

```
With Tree1
  With .VisualAppearance
    .Add &H20, App.Path + "\buttonu.ebn"
    .Add &H21, App.Path + "\buttond.ebn"
  End With
  .Background(exCellButtonUp) = &H20000000
  .Background(exCellButtonDown) = &H21000000
End With
```

The following C++ sample changes the appearance for buttons in the cells:

```
#include "Appearance.h"
```

```
m_tree.GetVisualAppearance().Add( 0x20,
COleVariant(_T("D:\\Temp\\ExTree.Help\\buttonu.ebn")) );
m_tree.GetVisualAppearance().Add( 0x21,
COleVariant(_T("D:\\Temp\\ExTree.Help\\buttond.ebn")) );
m_tree.SetBackground( 2 /*exCellButtonUp*/, 0x20000000 );
m_tree.SetBackground( 3 /*exCellButtonDown*/, 0x21000000 );
```

The following VB.NET sample changes the appearance for buttons in the cells.

```
With AxTree1
   With .VisualAppearance
      .Add(&H20, "D:\Temp\ExTree.Help\buttonu.ebn")
      .Add(&H21, "D:\Temp\ExTree.Help\buttond.ebn")
   End With
   .set_Background(EXTREELib.BackgroundPartEnum.exCellButtonUp, &H20000000)
   .set_Background(EXTREELib.BackgroundPartEnum.exCellButtonDown, &H21000000)
End With
```

The following C# sample changes the appearance for buttons in the cells.

```
axTree1.VisualAppearance.Add(0x20, "D:\\Temp\\ExTree.Help\\buttonu.ebn");
axTree1.VisualAppearance.Add(0x21, "D:\\Temp\\ExTree.Help\\buttond.ebn");
axTree1.set_Background(EXTREELib.BackgroundPartEnum.exCellButtonUp, 0x20000000);
axTree1.set_Background(EXTREELib.BackgroundPartEnum.exCellButtonDown,
0x21000000);
```

The following VFP sample changes the appearance for buttons in the cells.

```
With thisform.Tree1
   With .VisualAppearance
      .Add(32, "D:\Temp\ExTree.Help\buttonu.ebn")
      .Add(33, "D:\Temp\ExTree.Help\buttond.ebn")
   EndWith
   .Object.Background(2) = 536870912
   .Object.Background(3) = 553648128
endwith
```

the 536870912 indicates the 0x20000000 value in hexadecimal, and the 553648128 indicates the 0x21000000 value in hexadecimal.

The following VB sample sets the cells of the first column to be of button type, and displays a message if the button is clicked:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.Items.CellHasButton(Item, 0) = True
End Sub

Private Sub Tree1_CellButtonClick(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
   MsgBox "The cell of button type has been clicked"
End Sub
```

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

```
With Tree1.Items
   .CellHasButton(.FocusItem, 0) = True
End With
```

The following C++ sample assigns a button to the focused cell:

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

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

```
With AxTree1.Items
   .CellHasButton(.FocusItem, 0) = True
End With
```

The following C# sample assigns a button to the focused cell:

```
axTree1.Items.set_CellHasButton(axTree1.Items.FocusItem, 0, true);
```

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

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellHasButton(0,0) = .t.
```

```
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellHasCheckBox([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value indicating whether the cell has associated a checkbox or not.

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.
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 PartialCheck property to allow partial check feature within the column. Use the CheckImage property to change the check box appearance. Use the FilterType property on exCheck to filter for checked or unchecked items. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following sample enumerates the cells in the first column and assign a checkbox to all of them:

```
Dim h As Variant
Tree1.BeginUpdate
With Tree1.Items
For Each h In Tree1.Items
   .CellHasCheckBox(h, 0) = True
Next
End With
Tree1.EndUpdate
```

The same thing we can do using the Def property like follows:

```
With Tree1.Columns(0)
   .Def(exCellHasCheckBox) = True
```

```
End With
```

The following sample shows how how set the type of cells to radio type while adding new items:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.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 Tree1_CellStateChanged(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
   Debug.Print "The cell """ & Tree1.Items.CellCaption(Item, ColIndex) & """ has changed its state. The new state is " & IIf(Tree1.Items.CellState(Item, ColIndes) = 0, "Unchecked", "Checked")
End Sub
```

The following VB sample adds a checkbox to the focused cell:

```
With Tree1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
End With
```

The following C++ sample adds a checkbox to the focused cell:

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

The following C# sample adds a checkbox to the focused cell:

```
axTree1.Items.set_CellHasCheckBox(axTree1.Items.FocusItem, 0, true);
```

The following VB.NET sample adds a checkbox to the focused cell:

```
With AxTree1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
End With
```

The following VFP sample adds a checkbox to the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellHasCheckBox(0,0) = .t.
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellHasRadioButton([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value indicating whether the cell has associated a radio button or not.

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.
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 sets the radio type for all cells in the first column, and group all of them in the same radio group (1234):

```
Dim h As Variant
Tree1.BeginUpdate
With Tree1.Items
For Each h In Tree1.Items
   .CellHasRadioButton(h, 0) = True
   .CellRadioGroup(h, 0) = 1234
Next
End With
Tree1.EndUpdate
```

or

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.Items.CellHasRadioButton(Item, 0) = True
```

```
    Tree1.Items.CellRadioGroup(Item, 0) = 1234
End Sub
```

To find out the radio cell that is checked in the radio group 1234 you have to call: MsgBox Tree1.Items.CellCaption(, Tree1.Items.CellChecked(1234))

The following sample group all cells of the first column into a radio group, and display the cell's checked on the radio group when the state of a radio group has been changed:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
    Tree1.Items.CellHasRadioButton(Item, 0) = True
    Tree1.Items.CellRadioGroup(Item, 0) = 1234 ' The 1234 is arbirary and it represents the identifier for the radio group
End Sub

Private Sub Tree1_CellStateChanged(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
    Debug.Print "In the 1234 radio group the """ & Tree1.Items.CellCaption(, Tree1.Items.CellChecked(1234)) & """ is checked."
End Sub
```

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

```
With Tree1.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_tree.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 AxTree1.Items
```

```
    .CellHasRadioButton(.FocusItem, 0) = True
    .CellRadioGroup(.FocusItem, 0) = 1234
End With
```

The following C# sample assigns a radio button to the focused cell:

```
axTree1.Items.set_CellHasRadioButton(axTree1.Items.FocusItem, 0, true);
axTree1.Items.set_CellRadioGroup(axTree1.Items.FocusItem, 0, 1234);
```

The following VFP sample assigns a radio button to the focused cell:

```
with thisform.Tree1.Items
    .DefaultItem = .FocusItem
    .CellHasRadioButton(0,0) = .t.
    .CellRadioGroup(0,0) = 1234
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellHyperLink ([Item as Variant], [ColIndex as Variant]) as Boolean

Specifies whether the cell's is highlighted when the cursor mouse is over 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, or a string expression that indicates the column's caption.
Boolean	A boolean expression that indicates whether the cell is highlighted when the cursor is over the cell.

Use the CellHyperLink property to add hyperlink cells to your list/tree. Use the HyperLinkClick event to notify your application when a hyperlink cell is clicked. Use the CellForeColor property to specify the cell's foreground color. Use the HyperLinkColor property to specify the hyperlink color. Use the <a> anchor element to mark hyperlinks in HTML captions.

property Items.CellImage ([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets an Image that is displayed on the cell's area.

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 value that indicates the image index. 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 a cell. Use the CellImages property to assign multiple icons to a cell. Use the Images method to assign icons to the control at runtime. You can add images at design time by dragging a file to image editor of the control. The CellImage = 0 removes the cell's image. The collection of Images is 1 based. The CellImageClick event occurs when the cell's image is clicked. Use the ItemFromPoint property to retrieve the part of the control being clicked. Use the CellHasCheckBox property to add a check box to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the **<img>** HTML tag to insert icons inside the cell's caption, if the CellCaptionFormat property is exHTML. Use the FilterType property on exImage to filter items by icons. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample sets cell's image for the first column while new items are added (to run the sample make sure that control's images collection is not empty):

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
  Tree1.Items.CellImage(Item, 0) = 1
End Sub
```

The following VB 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.),

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
```

```
    Tree1.Items.CellImage(Item, 0) = 1
End Sub

Private Sub Tree1_CellImageClick(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long)
    Tree1.Items.CellImage(Item, ColIndex) = Tree1.Items.CellImage(Item, ColIndex) Mod 2 +
1
End Sub
```

The following C++ sample displays the first icon in the focused cell:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetCellImage( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ), 1 );
```

The following C# sample displays the first icon in the focused cell:

```
axTree1.Items.set_CellImage(axTree1.Items.FocusItem, 0, 1);
```

The following VB.NET sample displays the first icon in the focused cell:

```
With AxTree1.Items
    .CellImage(.FocusItem, 0) = 1
End With
```

The following VFP sample displays the first icon in the focused cell:

```
with thisform.Tree1.Items
    .DefaultItem = .FocusItem
    .CellImage(0,0) = 1
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

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.

The CellImages property assigns multiple icons to a cell. The CellImage property assign a single icon to the cell. Instead if multiple icons need to be assigned to a single cell you have to use the CellImages property. The CellImages property takes a list of additional icons and display them in the cell. The list is separated by ',' and should contain numbers that represent indexes to Images list collection. Use the ItemFromPoint property to retrieve the part of the control being clicked. Use the CellHasCheckBox property to add a check box to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the **<img>** HTML tag to insert icons inside the cell's caption, if the CellCaptionFormat property is exHTML. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample assigns the first and third icon to the cell:

```
With Tree1.Items
    .CellImages(.ItemByIndex(0), 1) = "1,3"
End With
```

The following VB sample displays the index of icon being clicked:

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim i As HITEM, h As HitTestInfoEnum, c As Long
  With Tree1
    i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
  End With
  If (i <> 0) Then
    If exHTCellIcon = (h And exHTCellIcon) Then
      Debug.Print "The index of icon being clicked is: " & (h And &HFFFF0000) / 65536
```

```
    End If
  End If
End Sub
```

The following VB sample changes the icon being clicked when the cell contains multiple icons, using the CellIImages property, or a single icon using the CellIImage property. We assume that the control contains at least two icons.

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim i As HITEM, h As HitTestInfoEnum, c As Long
  With Tree1
    i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
  End With
  If (i <> 0) Then
    If exHTCellIcon = (h And exHTCellIcon) Then
      Dim iImage As Long, j As Long
      iImage = (h And &HFFFF0000) / 65536
      With Tree1.Items
        If (Len(.CellIImages(i, c)) > 0) Then
          Dim cI() As String, nCI As String
          cI = Split(.CellIImages(i, c), ",")
          cI(iImage) = cI(iImage) Mod 2 + 1
          For j = LBound(cI) To UBound(cI)
            nCI = nCI + cI(j) + IIf(j < UBound(cI), ",", "")
          Next
          .CellIImages(i, c) = nCI
        Else
          .CellIImage(i, c) = .CellIImage(i, c) Mod 2 + 1
        End If
      End With
    End If
  End If
End Sub
```

The following C++ sample assigns the first and the third icon to the cell:

```
#include "Items.h"
CItems items = m_tree.GetItems();
```

```
items.SetCellImages( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ),
COleVariant( "1,3" ) );
```

The following C++ sample displays the index of icon being clicked:

```
#include "Items.h"
void OnMouseUpTree1(short Button, short Shift, long X, long Y)
{
   CItems items = m_tree.GetItems();
   long c = 0, hit = 0, h = m_tree.GetItemFromPoint( X, Y, &c, &hit);
   if ( h != 0 )
   {
      if ( ( hit & 0x44 /*exHTCellIcon*/ ) == 0x44 )
      {
         CString strFormat;
         strFormat.Format( "The index of icon being clicked is: %i\n", (hit >> 16) );
         OutputDebugString( strFormat );
      }
   }
}
```

The following VB.NET sample assigns the first and the third icon to the cell:

```
With AxTree1.Items
   .CellImages(.FocusItem, 0) = "1,3"
End With
```

The following VB.NET sample displays the index of icon being clicked:

```
Private Sub AxTree1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseUpEvent) Handles AxTree1.MouseUpEvent
   With AxTree1
      Dim i As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
      i = .get_ItemFromPoint(e.x, e.y, c, hit)
      If (Not (i = 0)) Then
         Debug.WriteLine("The index of icon being clicked is: " & (hit And &HFFFF0000) /
65536)
      End If
   End With
```

```
End Sub
```

The following C# sample assigns the first and the third icon to the cell:

```
axTree1.Items.set_CellImages(axTree1.Items.FocusItem, 0, "1,3");
```

The following C# sample displays the index of icon being clicked:

```
private void axTree1_MouseUpEvent(object sender,
AxEXTREELib._ITreeEvents_MouseUpEvent e)
{
   int c = 0;
   EXTREELib.HitTestInfoEnum hit;
   int i = axTree1.get_ItemFromPoint(e.x, e.y, out c, out hit);
   if ((i != 0))
   {
      if ((Convert.ToUInt32(hit) &
Convert.ToUInt32(EXTREELib.HitTestInfoEnum.exHTCellIcon)) ==
Convert.ToUInt32(EXTREELib.HitTestInfoEnum.exHTCellIcon))
      {
         string s = axTree1.Items.get_CellCaption(i, c).ToString();
         s = "Cell: " + s + ", Icon's Index: " + (Convert.ToUInt32(hit) >> 16).ToString();
         System.Diagnostics.Debug.WriteLine(s);
      }
   }
}
```

The following VFP sample assigns the first and the third icon to the cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellImages(0,0) = "1,3"
endwith
```

The following VFP sample displays the index of icon being clicked:

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

```
local c, hit
c = 0
hit = 0
with thisform.Tree1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 )
      if ( bitand( hit, 68 )= 68 )
         wait window nowait .Items.CellCaption( 0, c ) + " " + Str( Int((hit - 68)/65536) )
      endif
   endif
endwith
```

Add the code to the MouseUp, MouseMove or MouseDown event,

property Items.CellItalic([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value that indicates whether the cell's caption 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 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 Tree1.Items
   .CellItalic(.FocusItem, 0) = True
End With
```

The following C++ sample makes italic the focused cell:

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

The following C# sample makes italic the focused cell:

```
axTree1.Items.set_CellItalic(axTree1.Items.FocusItem, 0, true);
```

The following VB.NET sample makes italic the focused cell:

```
With AxTree1.Items
   .CellItalic(.FocusItem, 0) = True
End With
```

The following VFP sample makes italic the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellItalic( 0, 0 ) = .t.
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellItem (Cell as HCELL) as HITEM

Retrieves the handle of the item that is owner for a specfic cell.

Type	Description
Cell as HCELL	A long expression that indicates the handle of the cell.
HITEM	A long expression that indicates the handle of the item.

Use the CellItem property to retrieve the item's handle. Use the ItemCell property to gets 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 Tree1.Items
   .CellCaption(.FocusItem, 0) = "this"
   .CellCaption(, .ItemCell(.FocusItem, 0)) = "this"
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.

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 (merging all cells in the same 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 SplitCell property to split a cell.

You can merge the first three cells in the root item using any of the following methods:

```
With Tree1
  With .Items
    .CellMerge(.RootItem(0), 0) = Array(1, 2)
  End With
End With
```

```
With Tree1

```

```
   .BeginUpdate
   With .Items
      Dim r As Long
      r = .RootItem(0)
      .CellMerge(r, 0) = 1
      .CellMerge(r, 0) = 2
   End With
   .EndUpdate
End With
```

```
With Tree1
   .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 Tree1
   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 Tree1
   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 sample shows few methods to unmerge cells:

```
With Tree1
   With .Items
      .UnmergeCells .ItemCell(.RootItem(0), 0)
   End With
End With
```

```
With Tree1
   With .Items
      Dim r As Long
      r = .RootItem(0)
      .UnmergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1))
   End With
End With
```

```
With Tree1
   .BeginUpdate
   With .Items
      .CellMerge(.RootItem(0), 0) = -1
      .CellMerge(.RootItem(0), 1) = -1
      .CellMerge(.RootItem(0), 2) = -1
   End With
   .EndUpdate
End With
```

The following VB sample merges the first three cells in the focused item:

```
With Tree1.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_tree.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 AxTree1.Items
   .CellMerge(.FocusItem, 0) = 1
   .CellMerge(.FocusItem, 0) = 2
End With
```

The following C# sample merges the first three cells in the focused item:

```
axTree1.Items.set_CellMerge(axTree1.Items.FocusItem, 0, 1);
axTree1.Items.set_CellMerge(axTree1.Items.FocusItem, 0, 2);
```

The following VFP sample merges the first three cells in the focused item:

```
with thisform.Tree1.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.CellParent ([Item as Variant], [ColIndex as Variant]) as Variant

Retrieves the parent of an inner cell.

Type	Description
Item as Variant	A long expression that indicates the handle of the item where the cell is, or 0. If the Item parameter is 0, the ColIndex parameter must indicate the handle of the cell.
ColIndex as Variant	A long expression that indicates the index of the column where a cell is divided, or a long expression that indicates the handle of the cell being divided, if the Item parameter is missing or it is zero.
Variant	A long expression that indicates the handle of the parent cell.

Use the CellParent property to get the parent of the inner cell. The SplitCell method splits a cell in two cells (the newly created cell is called inner cell). Use the InnerCell property to get the inner cell. Use the CellItem property to get the item that's the owner of the cell. The CellParent property gets 0 if the cell is not an inner cell. The parent cell is always displayed to the left side of the cell. The inner cell (InnerCell) is displayed to the right side of the cell.

The following VB sample determines whether the cell is a master cell or an inner cell:

```
Private Function isMaster(ByVal g As EXTREELibCtl.Tree, ByVal h As EXTREELibCtl.HITEM,
ByVal c As Long) As Boolean
   With g.Items
      isMaster = .CellParent(h, c) = 0
   End With
End Function
```

The following VB sample determines the master cell (the cell from where the splitting starts):

```
Private Function getMaster(ByVal g As EXTREELibCtl.Tree, ByVal h As EXTREELibCtl.HITEM,
ByVal c As Long) As EXTREELibCtl.HCELL
   With g.Items
      Dim r As EXTREELibCtl.HCELL
      r = c
      If Not (h = 0) Then
         r = .ItemCell(h, c)
```

```
    End If
    While Not (.CellParent(, r) = 0)
      r = .CellParent(, r)
    Wend
    getMaster = r
  End With
End Function
```

The following C++ sample determines whether the cell is a master cell or an inner cell:

```
#include "Items.h"

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

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

BOOL isMaster( CTree tree, long hItem, long nColIndex )
{
  return V2I( &tree.GetItems().GetCellParent( COleVariant( hItem ), COleVariant( nColIndex
) ) ) == 0;
}
```

The following C++ sample determines the master cell (the cell from where the splitting starts):

```
long getMaster( CTree tree, long hItem, long nColIndex )
{
  COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
```

```
    CItems items = tree.GetItems();
    long r = nColIndex;
    if ( hItem )
        r = items.GetItemCell( hItem, COleVariant( nColIndex ) );
    long r2 = 0;
    while ( r2 = V2I( &items.GetCellParent( vtMissing, COleVariant( r ) ) ) )
        r = r2;
    return r;
}
```

The following VB.NET sample determines whether the cell is a master cell or an inner cell:

```
Private Function isMaster(ByVal g As AxEXTREELib.AxTree, ByVal h As Long, ByVal c As
Long) As Boolean
    With g.Items
        isMaster = .CellParent(h, c) = 0
    End With
End Function
```

The following VB.NET sample determines the master cell (the cell from where the splitting starts):

```
Shared Function getMaster(ByVal g As AxEXTREELib.AxTree, ByVal h As Integer, ByVal c As
Integer) As Integer
    With g.Items
        Dim r As Integer
        r = c
        If Not (h = 0) Then
            r = .ItemCell(h, c)
        End If
        While Not (.CellParent(, r) = 0)
            r = .CellParent(, r)
        End While
        getMaster = r
    End With
End Function
```

The following C# sample determines whether the cell is a master cell or an inner cell:

```
private bool isMaster(AxEXTREELib.AxTree tree, int h, int c)
{
   return Convert.ToInt32(tree.Items.get_CellParent(h, c)) != 0;
}
```

The following C# sample determines the master cell (the cell from where the splitting starts):

```
private long getMaster(AxEXTREELib.AxTree g, int h, int c)
{
   int r = c, r2 = 0;
   if ( h != 0 )
      r = Convert.ToInt32( g.Items.get_ItemCell(h,c) );
   r2 = Convert.ToInt32( g.Items.get_CellParent(null, r));
   while ( r2 != 0)
   {
      r = r2;
      r2 = Convert.ToInt32( g.Items.get_CellParent(null, r));
   }
   return r;
}
```

property Items.CellPicture ([Item as Variant], [ColIndex as Variant]) as Variant

Retrieves or sets a value that indicates the Picture object displayed by 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 Picture object that indicates the cell's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format.

The control can associate to a cell a check or radio button, an icon, multiple icons, a picture and a caption. Use the CellPicture property to associate a picture to a cell. You can use the CellPicture property when you want to display images with different widths into a cell. Use the CellImage property to associate an icon from Images collection. Use the CellImages property to assign multiple icons to a cell. Use the CellHasCheckBox property to add a check box to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. The CellPictureWidth and CellPictureHeight properties specifies the size of the area where the cell's picture is stretched.

The following VB sample loads a picture from a file:

```
Tree1.Items.CellPicture(h, 0) = LoadPicture("c:\winnt\logo.gif")
```

The following VB sample associates a picture to a cell by loading it from a base64 encoded string:

```
Dim s As String
s =
"gBCJr+BAAg0HGwEgwog4jg4ig4BAEFg4AZEKisZjUbAAzg5mg6Zg7Mg7/g0ek8oGcgjsijsk

s = s +
"XgBadIDXdYSXRb9wWBclK2taF1gAI5HiPaN8oPdlNWbaF23KAwyWkNYyXxg9p3WNYjU/c

With Tree1
   .BeginUpdate
```

```
   .Columns.Add "Column 1"
   With .Items
      Dim h As HITEM
      h = .AddItem("Item 1")
      .CellPicture(h, 0) = s
      .ItemHeight(h) = 24
   End With
   .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;
               _ASSERTE( dwFileSize == dwReadBytes );
               if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal, TRUE, &spStream;) ) )
                  if ( SUCCEEDED( hResult = OleLoadPicture( spStream, 0, FALSE,
IID_IPictureDisp, (void**)ppPictureDisp ) ) )
                     bResult = TRUE;
            }
         }
         CloseHandle( hFile );
      }
   }
   return bResult;
}

IPictureDisp* pPicture = NULL;
if ( LoadPicture( "c:\\winnt\\zapotec.bmp", &pPicture; ) )
{
   COleVariant vtPicture;
   V_VT( &vtPicture; ) = VT_DISPATCH;
   pPicture->QueryInterface( IID_IDispatch, (LPVOID*)&V;_DISPATCH( &vtPicture; ) );
   CItems items = m_tree.GetItems();
   items.SetCellPicture( COleVariant( items.GetFocusItem() ), COleVariant(long(0)), vtPicture
);
   pPicture->Release();
}
```

The following VB.NET sample loads a picture from a file:

```
With AxTree1.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:

```
axTree1.Items.set_CellPicture(axTree1.Items.FocusItem, 0,
IPDH.GetIPictureDisp(Image.FromFile("c:\\winnt\\zapotec.bmp")));
```

where the IPDH class is defined like follows:

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

    public static object GetIPictureDisp(System.Drawing.Image image)
    {
        return System.Windows.Forms.AxHost.GetIPictureDispFromPicture( image );
    }
}
```

The following VFP sample loads a picture from a file:

```
with thisform.Tree1.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 the radio group where the cell is contained.

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 value 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 sets the radio type for all cells in the first column, and group all of them in the same radio group (1234):

```
Dim h As Variant
Tree1.BeginUpdate
With Tree1.Items
For Each h In Tree1.Items
   .CellHasRadioButton(h, 0) = True
   .CellRadioGroup(h, 0) = 1234
Next
End With
Tree1.EndUpdate
```

or

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.Items.CellHasRadioButton(Item, 0) = True
   Tree1.Items.CellRadioGroup(Item, 0) = 1234
End Sub
```

To find out the radio cell that is checked in the radio group 1234 you have to call: MsgBox

Tree1.Items.CellCaption(, Tree1.Items.CellChecked(1234))

The following sample group all cells of the first column into a radio group, and display the cell's checked on the radio group when the state of a radio group has been changed:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.Items.CellHasRadioButton(Item, 0) = True
   Tree1.Items.CellRadioGroup(Item, 0) = 1234 ' The 1234 is arbirary and it represents the identifier for the radio group
End Sub

Private Sub Tree1_CellStateChanged(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
   Debug.Print "In the 1234 radio group the """ & Tree1.Items.CellCaption(, Tree1.Items.CellChecked(1234)) & """ is checked."
End Sub
```

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

```
With Tree1.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_tree.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 AxTree1.Items
   .CellHasRadioButton(.FocusItem, 0) = True
   .CellRadioGroup(.FocusItem, 0) = 1234
End With
```

The following C# sample assigns a radio button to the focused cell:

```
axTree1.Items.set_CellHasRadioButton(axTree1.Items.FocusItem, 0, true);
axTree1.Items.set_CellRadioGroup(axTree1.Items.FocusItem, 0, 1234);
```

The following VFP sample assigns a radio button to the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellHasRadioButton(0,0) = .t.
   .CellRadioGroup(0,0) = 1234
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

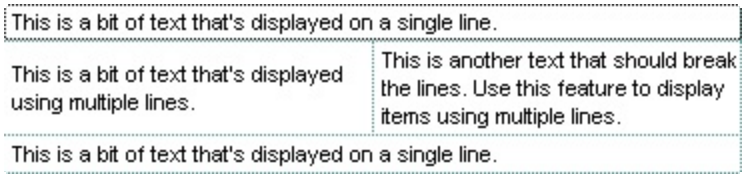
```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellSingleLine([Item as Variant], [ColIndex as Variant]) as CellSingleLineEnum

Retrieves or sets a value indicating whether the cell's caption is painted using one or more lines.

Type	Description
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.
CellSingleLineEnum	A CellSingleLineEnum expression that specifies how the caption breaks

By default, the CellSingleLine property is exCaptionSingleLine / True, which indicates that the cell's caption is displayed on a single line. Use the Def(exCellSingleLine) property to specify that all cells in the column display their content using multiple lines. The control can displays the cell's caption using more lines, if the CellSingleLine property is exCaptionWordWrap or exCaptionBreakWrap. The CellSingleLine property wraps the cell's caption so it fits in the cell's client area. If the text doesn't fit the cell's client area, the height of the item is increased or decreased. When the CellSingleLine is exCaptionWordWrap / exCaptionBreakWrap / False, the height of the item is computed based on each cell caption. *If the CellSingleLine property is* exCaptionWordWrap / exCaptionBreakWrap / *False, changing the ItemHeight property has no effect.* Use the ItemMaxHeight property to specify the maximum height of the item when its height is variable. Use the CellVAlignment property to align vertically a cell.



If using the CellSingleLine / Def(exCellSingleLine) property, we recommend to set the ScrollBySingleLine property on True so all items can be scrolled.

The following VB sample displays the caption of the focused cell using multiple lines:

```
With Tree1.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_tree.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 AxTree1.Items
   .CellSingleLine(.FocusItem, 0) = False
End With
```

The following C# sample displays the caption of the focused cell using multiple lines:

```
axTree1.Items.set_CellSingleLine(axTree1.Items.FocusItem, 0, false);
```

The following VFP sample displays the caption of the focused cell using multiple lines:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellSingleLine( 0, 0 ) = .f.
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellState([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets the cell's state. Has effect only for check and radio cells.

Type	Description
Item as Variant	A long expression that indicates the item's handle that indicates the owner of the cell.
ColIndex as Variant	A long expression that identifies the column's index, or a string expression that specifies 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 PartialCheck property to allow partial check feature within the column. Use the CheckImage property to change the check box appearance. Use the RadioImage property to change the radio button appearance. Use the FilterType property on exCheck to filter for checked or unchecked items.

The following VB sample adds a check box that's checked to the focused cell:

```
With Tree1.Items
   .CellHasCheckBox(.FocusItem, 0) = True
   .CellState(.FocusItem, 0) = 1
End With
```

The following C++ sample adds a check box that's checked to the focused cell:

```
#include "Items.h"
CItems items = m_tree.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 AxTree1.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:

```
axTree1.Items.set_CellHasCheckBox(axTree1.Items.FocusItem, 0, true);
axTree1.Items.set_CellState(axTree1.Items.FocusItem, 0, 1);
```

The following VFP sample adds a check box that's checked to the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellHasCheckBox( 0, 0 ) = .t.
   .CellState( 0,0 ) = 1
endwith
```

The following VB sample changes the state for a cell to checked state:
Tree1.Items.CellState(Tree1.Items(0), 0) = 1,

The following VB sample changes the state for a cell to to unchecked state:
Tree1.Items.CellState(Tree1.Items(0), 0) = 0,

The following VB sample changes the state for a cell to partial checked state:
Tree1.Items.CellState(Tree1.Items(0), 0) = 2

The following VB sample displays a message when a cell of radio or check type is changing its state:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.Items.CellHasCheckBox(Item, 0) = True
End Sub

Private Sub Tree1_CellStateChanged(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
   Debug.Print "The cell """ & Tree1.Items.CellCaption(Item, ColIndex) & """ has changed its state. The new state is " & Iif(Tree1.Items.CellState(Item, ColIndes) = 0, "Unchecked", "Checked")
End Sub
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see

Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellStrikeOut([Item as Variant], [ColIndex as Variant]) as Boolean

Retrieves or sets a value that indicates 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 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's caption 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.

The following VB sample draws a horizontal line through the caption of the cell that has the focus:

```
With Tree1.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_tree.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:

```
axTree1.Items.set_CellStrikeOut(axTree1.Items.FocusItem, 0, true);
```

The following VB.NET sample draws a horizontal line through the caption of the cell that has the focus:

```
With AxTree1.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.Tree1.Items
   .DefaultItem = .FocusItem
   .CellStrikeOut(0, 0 ) = .t.
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

property Items.CellToolTip([Item as Variant], [ColIndex as Variant]) as String

Retrieves or sets a text that is used to show the tooltip's cell.

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 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 not empty and different than "...", the CellToolTip property indicates the description of the cell's tooltip. 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. Use the ToolTipFont property to assign a font for the control's tooltip. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the ShowToolTip method to display a custom tooltip.

The tooltip supports the following HTML tags:

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

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

the user clicks/collapses/expands the caption.

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

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

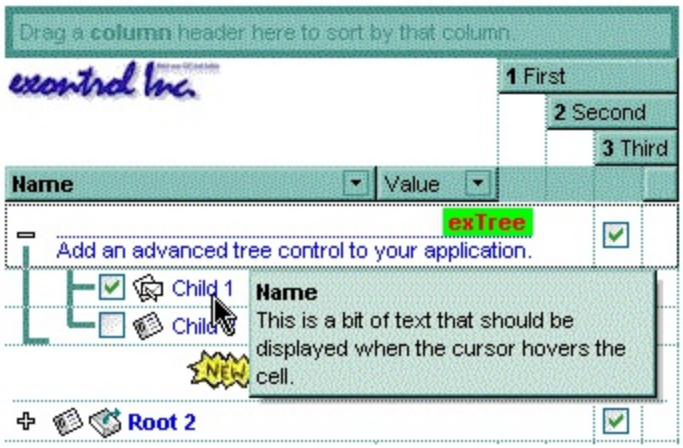
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays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

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

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

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

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



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's caption should appear in underline.

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.
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 Tree1.Items
   .CellUnderline(.FocusItem, 0) = True
End With
```

The following C++ sample underlines the focused cell:

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

The following C# sample underlines the focused cell:

```
axTree1.Items.set_CellUnderline(axTree1.Items.FocusItem, 0, true);
```

The following VB.NET sample underlines the focused cell:

```
With AxTree1.Items
   .CellUnderline(.FocusItem, 0) = True
```

```
End With
```

The following VFP sample underlines the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellUnderline(0, 0 ) = .t.
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

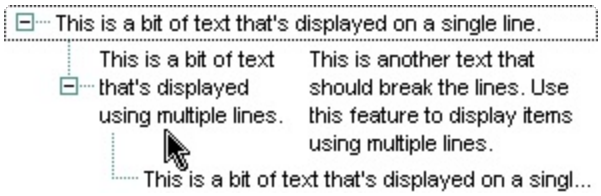
```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

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 identifies the item's handle
ColIndex as Variant	A long expression that indicates the column's index or the cell's handle, a string expression that indicates the column's caption.
VAlignmentEnum	A VAlignmentEnum expression that indicates the cell's vertically alignment.

Use the CellVAlignment property to specify the vertically alignment for the cell's caption. Use the CellSingleLine property to specify whether a cell uses single or multiple lines. 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 Tree1.Items
   .CellVAlignment(.FocusItem, 0) = VAlignmentEnum.BottomAlignment
End With
```

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

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetCellVAlignment( COleVariant( items.GetFocusItem() ), COleVariant( (long)0 ), 2
/*BottomAlignment*/ );
```

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

```
With AxTree1.Items
   .CellVAlignment(.FocusItem, 0) = EXTREELib.VAlignmentEnum.BottomAlignment
```

```
End With
```

The following C# sample right aligns the focused cell:

```
axTree1.Items.set_CellVAlignment(axTree1.Items.FocusItem, 0,
EXTREELib.VAlignmentEnum.BottomAlignment);
```

The following VFP sample right aligns the focused cell:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .CellVAlignment(0,0) = 2 && BottomAlignment
endwith
```

property Items.CellWidth([Item as Variant], [ColIndex as Variant]) as Long

Retrieves or sets a value that indicates the width of the inner cell.

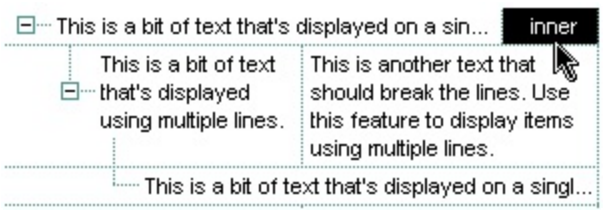
Type	Description
Item as Variant	A long expression that indicates the handle of the item where the cell is, or 0. If the Item parameter is 0, the ColIndex parameter must indicate the handle of the cell.
ColIndex as Variant	A long expression that indicates the index of the column where a cell is divided, or a long expression that indicates the handle of the cell being divided, if the Item parameter is missing or it is zero.
Long	A long expression that indicates the width of the cell.

The CellWidth property specifies the cell's width. The CellWidth property has effect only if the cell contains inner cells. The SplitCell method splits a cell in two cells (the newly created cell is called inner cell). Use the InnerCell property to get the inner cell. Use the CellParent property to get the parent of the inner cell. Use the CellItem property to get the item that's the owner of the cell. Use the BeginUpdate and EndUpdate methods to refresh the cell's width when changing it on the fly.

The CellWidth property specifies the width of the cell, where the cell is divided in two or multiple (inner) cells like follows:

- if the CellWidth property is less than zero, the master cell calculates the width of the inner cell, so all the inner cells with CellWidth less than zero have the same width in the master cell.
- if the CellWidth property is greater than zero, it indicates the width in pixels of the inner cell.

By default, the CellWidth property is -1, and so when the user splits a cell the inner cell takes the right half of the area occupied by the master cell.



The following VB sample splits the first visible cell in three cells:

```
With Tree1
  .BeginUpdate
  .DrawGridLines = exAllLines
```

```
  With .Items
    Dim h As HITEM, f As HCELL
    h = .FirstVisibleItem
    f = .ItemCell(h, 0)
    f = .SplitCell(, f)
    .CellCaption(, f) = "Split 1"
    f = .SplitCell(, f)
    .CellCaption(, f) = "Split 2"
  End With
  .EndUpdate
End With
```

The following VB sample specifies that the inner cell should have 32 pixels:

```
With Tree1
  .BeginUpdate
  .DrawGridLines = exAllLines
  With .Items
    Dim h As HITEM, f As HCELL
    h = .FirstVisibleItem
    f = .ItemCell(h, 0)
    f = .SplitCell(, f)
    .CellCaption(, f) = "Split"
    .CellWidth(, f) = 32
  End With
  .EndUpdate
End With
```

The following VB sample adds an inner cell to the focused cell with 48 pixels width:

```
Tree1.BeginUpdate
With Tree1.Items
  Dim h As Long
  h = .SplitCell(.FocusItem, 0)
  .CellBackColor(, h) = vbBlack
  .CellForeColor(, h) = vbWhite
  .CellHAlignment(, h) = CenterAlignment
  .CellCaption(, h) = "inner"
```

```
    .CellWidth(, h) = 48
End With
Tree1.EndUpdate
```

The following C++ sample adds an inner cell to the focused cell with 48 pixels width:

```
#include "Items.h"
m_tree.BeginUpdate();
CItems items = m_tree.GetItems();
COleVariant vtItem( items.GetFocusItem() ), vtColumn( long(0) ), vtMissing; V_VT(
&vtMissing ) = VT_ERROR;
COleVariant vtInner = items.GetSplitCell( vtItem, vtColumn );
items.SetCellWidth( vtMissing, vtInner, 48 );
items.SetCellBackColor( vtMissing, vtInner, 0 );
items.SetCellForeColor( vtMissing, vtInner, RGB(255,255,255) );
items.SetCellCaption( vtMissing, vtInner, COleVariant("inner") );
items.SetCellHAlignment( vtMissing, vtInner, 1 );
m_tree.EndUpdate();
```

The following VB.NET sample adds an inner cell to the focused cell with 48 pixels width:

```
With AxTree1
    .BeginUpdate()
    With .Items
        Dim iInner As Integer
        iInner = .SplitCell(.FocusItem, 0)
        .CellCaption(, iInner) = "inner"
        .CellHAlignment(, iInner) = EXTREELib.AlignmentEnum.CenterAlignment
        .CellWidth(, iInner) = 48
        .CellBackColor(, iInner) = 0
        .CellForeColor(, iInner) = ToUInt32(Color.White)
    End With
    .EndUpdate()
End With
```

The following C# sample adds an inner cell to the focused cell with 48 pixels width:

```
EXTREELib.Items items = axTree1.Items;
axTree1.BeginUpdate();
```

```
object iInner = items.get_SplitCell(axTree1.Items.FocusItem, 0);
items.set_CellCaption(null, iInner, "inner");
items.set_CellHAlignment(null, iInner, EXTREELib.AlignmentEnum.CenterAlignment);
items.set_CellBackColor(null, iInner, ToUInt32(Color.Black));
items.set_CellForeColor(null, iInner, ToUInt32(Color.White));
items.set_CellWidth(null, iInner, 48);
axTree1.EndUpdate();
```

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 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 the cell's handle, a string expression that indicates the column's caption.

The ClearCellBackColor method clears the cell's background color when the CellBackColor property is 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 the cell's handle, a string expression that indicates the column's caption.

The ClearCellForeColor method clears the cell's foreground color when CellForeColor property was used.

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 HITEM	A long expression that indicates the item's handle. If the Item is 0, the ClearItemBackColor clears the background color for all items.

The ClearItemBackColor method clears the item's background color when ItemBackColor property is used.

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 is used. Use the ForeColor property to change the control's foreground color.

property Items.DefaultItem as HITEM

Retrieves or sets the default item's handle.

Type	Description
HITEM	A long expression that indicates the handle of the item that's used by all properties of the 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 grater 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, and the VFP thinks that the CellImage is an array, but it is not. It is a property. Hope that future versions will correct this problem in VFP.

So, if you pass zero to a property that has a parameter titled Item, the control takes instead the DefaultItem value.

method Items.Edit ([Item as Variant], [ColIndex as Variant])

Edits a 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.

The Edit method starts editing an item. The edit operation starts only if the control's AllowEdit property is True. When the edit operation starts the control fires the BeforeCellEdit event. Use the BeforeCellEdit event to cancel the edit operation. When the edit operation ends the control fires the AfterCellEdit event. Use the AfterCellEdit event to change the cell's caption after edit operation ends. Use the SelStart, SelLength properties to specify the coordinates of the text being selected when edit starts. The following code starts editing the first cell: Tree1.Items.Edit Tree1.Items(0), 0.

The following VB sample changes the cell's caption when the edit operation ends:

```
Private Sub Tree1_AfterCellEdit(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long, ByVal NewCaption As String)
   Tree1.Items.CellCaption(Item, ColIndex) = NewCaption
End Sub
```

The following VB sample starts editing the cell as soon as the user clicks the item:

```
Private Sub Tree1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   ' 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 EXTREELibCtl.HitTestInfoEnum
   ' Gets the item from (X,Y)
   h = Tree1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Then
      With Tree1
         .AllowEdit = True
         With .Items
```

```
            .Edit h, 0
        End With
    End With
  End If
End Sub
```

The following VB.NET sample changes the cell's caption as soon as the edit operation ends.

```
Private Sub AxTree1_AfterCellEdit(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_AfterCellEditEvent) Handles AxTree1.AfterCellEdit
  AxTree1.Items.CellCaption(e.item, e.colIndex) = e.newCaption
End Sub
```

The following C# sample changes the cell's caption as soon as the edit operation ends.

```
private void axTree1_AfterCellEdit(object sender,
AxEXTREELib._ITreeEvents_AfterCellEditEvent e)
{
  axTree1.Items.set_CellCaption( e.item, e.colIndex, e.newCaption );
}
```

The following C++ sample changes the cell's caption as soon as the edit operation ends.

```
void OnAfterCellEditTree1(long Item, long ColIndex, LPCTSTR NewCaption)
{
  m_tree.GetItems().SetCellCaption( COleVariant( Item ), COleVariant( ColIndex ),
COleVariant( NewCaption ) );
}
```

The following VFP sample changes the cell's caption as soon as the edit operation ends.

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, newcaption

with thisform.Tree1.Items
  .DefaultItem = item
  .CellCaption( 0, colindex ) = newcaption
endwith
```

Note: A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

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

Use the EnableItem property to disable an item. A disabled item looks grayed and it is selectable. Use the SelectableItem property to specify the user can select an item. Once that an item is disabled all the cells of the item are disabled, so CellEnabled property has no effect. To disable a column you can use Enabled property of a Column object.

method Items.EnsureVisibleItem (Item as HITEM)

Ensures the given item is in the visible client area.

Type	Description
Item as HITEM	A long expression that indicates the item's handle that fits the client area.

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. Use the Scroll method to scroll programmatically the control. Use the EnsureVisibleColumn method to ensure that a specified column fits the control's client area.

The following VB sample ensures that first item is visible:

```
Tree1.Items.EnsureVisibleItem Tree1.Items(0)
```

The following C++ sample ensures that first item is visible:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.EnsureVisibleItem( items.GetItemByIndex( 0 ) );
```

The following C# sample ensures that first item is visible:

```
axTree1.Items.EnsureVisibleItem(axTree1.Items[0]);
```

The following VB.NET sample ensures that first item is visible:

```
AxTree1.Items.EnsureVisibleItem( AxTree1.Items.FocusItem );
```

The following VFP sample ensures that first item is visible:

```
with thisform.Tree1.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 handle of the item being expanded or collapsed. 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 items is expanded or collapsed. Before expanding/collapsing an item, the control fires the BeforeExpandItem event. Use the BeforeExpandIvent to cancel expanding/collapsing of an item. After item was expanded/collapsed the control fires the AfterExpandItem event. The following samples shows how to expand the selected item: Tree1.Items.ExpandItem(Tree1.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 ExpandOnKeys property to specify whether the user expands or collapses the focused items using arrow keys. Use the InsertItem property to add child items.

The following VB sample programmatically expands the item when the user selects it :

```
Private Sub Tree1_SelectionChanged()
   Tree1.Items.ExpandItem(Tree1.Items.SelectedItem()) = True
End Sub
```

The following VB sample expands programmatically the focused item:

```
With Tree1.Items
   .ExpandItem(.FocusItem) = True
End With
```

The following C++ sample expands programmatically the focused item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetExpandItem( items.GetFocusItem(), TRUE );
```

The following VB.NET sample expands programmatically the focused item:

```
AxTree1.Items.ExpandItem( AxTree1.Items.FocusItem ) = True
```

The following C# sample expands programmatically the focused item:

```
axTree1.Items.set_ExpandItem( axTree1.Items.FocusItem, true );
```

The following VFP sample expands programmatically the focused item:

```
with thisform.Tree1.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 column. The searching starts at StartIndex item.

Type	Description
Caption as Variant	A Variant expression that indicates the caption that is searched for.
ColIndex as Variant	A long expression that indicates the column's index or the cell's handle, a string expression that indicates the column's caption.
StartIndex as Variant	A long value that indicates the index of item from where the searching starts. Use the ItemToIndex property to convert the handle of the item to an index.
HITEM	A long expression that indicates the item's handle that matches the criteria. Use the ItemToIndex property to convert the handle of the item to an index

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 searching within the column.

The following VB sample selects the first item that matches "DUMON" on the first column:

```
Tree1.Items.SelectItem(Tree1.Items.FindItem("DUMON", 0)) = True
```

The following C++ sample finds and selects an item:

```
#include "Items.h"
CItems items = m_tree.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:

```
axTree1.Items.set_SelectItem(axTree1.Items.get_FindItem("Child 2", 0, 0), true);
```

The following VB.NET sample finds and selects an item:

```
With AxTree1.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.Tree1.Items
   .DefaultItem = .FindItem("Child 2",0)
   if ( .DefaultItem <> 0 )
      .SelectItem( 0 ) = .t.
   endif
endwith
```

property Items.FindItemData (UserData as Variant, [StartIndex as Variant]) as HITEM

Finds the item giving its data.

Type	Description
UserData as Variant	A Variant expression that indicates the value being searched.
StartIndex as Variant	A long expression that indicates the index of the item where the searching starts.
HITEM	A long expression that indicates the handle of the item found.

Use the FindItemData property to search for an item giving its extra-data. Use the ItemData property to associate an extra data to an item. Use the FindItem property to locate an item given its caption. Use the FindPath property to search for an item given its path.

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. The searching is case sensitive only if the ASCIIUpper property is empty. 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:

```
Tree1.Items.SelectItem(Tree1.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_tree.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 AxTree1.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 = axTree1.Items.get_FindPath("Files and Folders\\Hidden Files and Folders\\Do not show hidden files and folder");
if ( iFind != 0 )
   axTree1.Items.set_SelectItem(iFind, true);
```

The following VFP sample selects the item based on its path:

```
with thisform.Tree1.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 into 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 the items that fit the control's client area:

```
On Error Resume Next
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = Tree1.Columns.Count
With Tree1.Items
   h = .FirstVisibleItem
   While Not (h = 0) And .IsItemVisible(h)
      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 the items that fit the control's client area:

```
#include "Items.h"
CItems items = m_tree.GetItems();
long hItem = items.GetFirstVisibleItem();
while ( hItem && items.GetIsItemVisible( hItem ) )
{
   OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
```

```
long(0) ) ) ) );
   hItem = items.GetNextVisibleItem( hItem );
}
```

The following VB.NET sample enumerates the items that fit the control's client area:

```
With AxTree1.Items
   Dim hItem As Integer
   hItem = .FirstVisibleItem
   While Not (hItem = 0)
      If (.IsItemVisible(hItem)) Then
         Debug.Print(.CellCaption(hItem, 0))
         hItem = .NextVisibleItem(hItem)
      Else
         Exit While
      End If
   End While
End With
```

The following C# sample enumerates the items that fit the control's client area:

```
EXTREELib.Items items = axTree1.Items;
int hItem = items.FirstVisibleItem;
while ( ( hItem != 0 ) && (items.get_IsItemVisible(hItem)) )
{
   object strCaption = items.get_CellCaption(hItem, 0);
   System.Diagnostics.Debug.WriteLine( strCaption != null ? strCaption.ToString() : "" );
   hItem = items.get_NextVisibleItem(hItem);
}
```

The following VFP sample enumerates the items that fit the control's client area:

```
with thisform.Tree1.Items
   .DefaultItem = .FirstVisibleItem
   do while ( ( .DefaultItem <> 0 ) and ( .IsItemVisible( 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 SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. Use the ShowFocusRect property to indicate whether the control draws a marking rectangle around the focused item. You can change the focused item, by selecting a new item using the SelectItem method. If the items is not selectable, it is not focusable as well. Use the SelectableItem property to specify whether an item is selectable/focusable.

property Items.FormatCell([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:

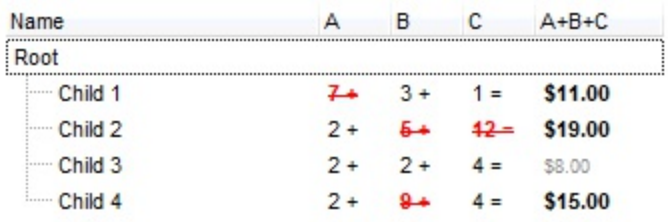
- formatted using the FormatCell property, if it is valid
- formatted using the FormatColumn property, if it is valid

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 "**M**ihai Filimon".

- the "*len(value) ? ((0:=dbl(value)) < 10 ? '<fgcolor=808080><font ;7>' : '<b>') + 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):



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

The expression supports cell's identifiers as follows:

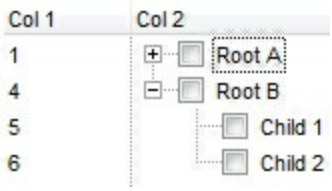
- ***%0**, **%1**, **%2**, ...{any} specifies the value of the cell in the column with the index 0, 1 2, ... The CellCaption property defines the cell's value. For example, "%0 format ``" formats the value in the cell at index 0 using the current regional settings, while "int(%1)" converts the value in the cell at index 1 to an integer.*
- ***%C0**, **%C1**, **%C2**, ...{string} specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property gets the cell's formatted caption. The cell's displayed string may differ from its actual value. For example, if a cell displays HTML content, %0 returns the HTML format including the tags, while %C0 returns the cell's content as a plain string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell at index 1 to uppercase, while "%C0 left 2" returns the leftmost two characters of the caption in the cell at index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ...{any} specifies the cell's extra data in the column with the index 0, 1 2, ... The CellData property associates any extra/user data to a cell. For example, "%CD0 = your user data" specifies all cells in the column with index 0 whose CellData property is equal to your user data.*
- ***%CS0**, **%CS1**, **%CS2**, ...{number} specifies the cell's state in the column with the index 0, 1 2, ... The CellState property defines the state of a cell, indicating whether it is checked or unchecked. For example, "%CS0" identifies all checked items in the column with index 0, while "not %CS1" identifies all unchecked items in the column with index 1.*
- ***%CC0**, **%CC1**, **%CC2**, ... {number} retrieve the number of child items (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ChildCount property returns the number of child items. For example, "%CC0" identifies all parent items, while "%CC0 = 0" identifies all leaf items.*
- ***%CX0**, **%CX1**, **%CX2**, ... {boolean} returns true if the item hosting the cell is expanded, or false if it is collapsed (this keyword consistently returns identical results for all cells since it pertains to the item that hosts each cell). The ExpandItem property specifically indicates whether the item is expanded or collapsed. For example,*

"%CX0" refers to all expanded items, while "not %CX0" identifies all collapsed items

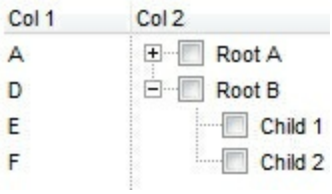
The predefined operators for auto-numbering are:

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

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

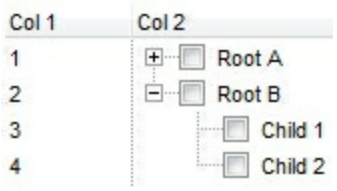


  In the following screen shot the FormatColumn("Col 1") = "1 index 'A-Z'"

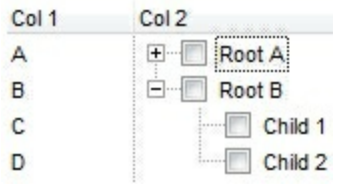


- number **apos** 'format' indicates the absolute position of the item. The first displayed item has the absolute position 0 (scrolling position on top), the next visible item is 1, and so on. The number indicates the starting position, while the format is a set of characters to be used for specifying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

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

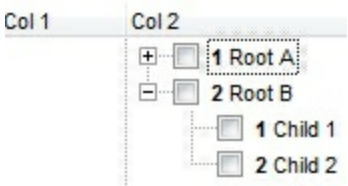


In the following screen shot the FormatColumn("Col 1") = "1 apos 'A-Z'"

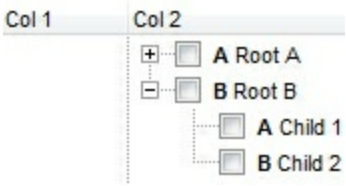


- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance,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

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



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

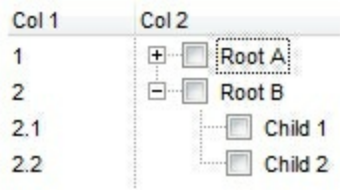


- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position.For instance, 1 pos '' gets the relative position of the item starting from 1, while 100 pos '' gets the relative position of the item starting from 100.

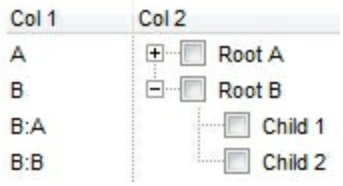
If the format is missing, the relative position of the item is formatted as numbers. *The difference between pos and opos can be seen while filtering the items in the control. For instance, if no filter is applied to the control, the pos and opos gets the same result. Instead, if the filter is applied,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

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

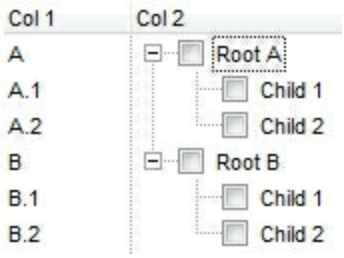
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ""



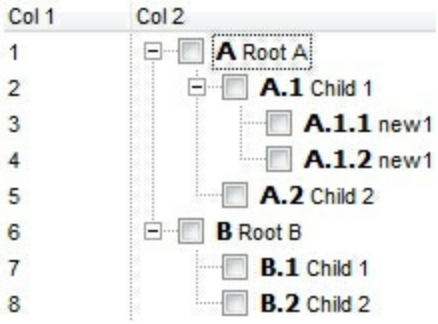
  In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"



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



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



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

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

property Items.FullPath (Item as HITEM) as String

Returns the fully qualified path of the referenced item in the ExTree 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.

property Items.InnerCell ([Item as Variant], [ColIndex as Variant], [Index as Variant]) as Variant

Retrieves the inner cell.

Type	Description
Item as Variant	A long expression that indicates the handle of the item where the cell is, or 0. If the Item parameter is 0, the ColIndex parameter must indicate the handle of the cell.
ColIndex as Variant	A long expression that indicates the index of the column where a cell is divided, or a long expression that indicates the handle of the cell being divided, if the Item parameter is missing or it is zero.
Index as Variant	A long expression that indicates the index of the inner being requested. If the Index parameter is missing or it is zero, the InnerCell property retrieves the master cell.
Variant	A long expression that indicates the handle of the inner cell.

Use the InnerCell property to get the inner cell. The InnerCell(, , 0) property always retrieves the same cell. The InnerCell(, , 1) retrieves the first inner cell, and so on. The InnerCells property always retrieves a non empty value. For instance, if a cell contains only two splited cells, the InnerCell(, , 3), or InnerCell(, , 4), and so on, always retrieves the last inner cell. The SplitCell method splits a cell in two cells (the newly created cell is called inner cell). Use the CellParent property to get the parent of the inner cell. Use the CellItem property to get the item that's the owner of the cell. Use the CellWidth property to specify the width of the inner cell. Use the CellParent property to determine whether the cell is a master cell or an inner cell. If the CellParent property gets 0, it means that the cell is master, else it is inner.

The following VB sample specifies whether a cell contains inner cells (the function checks whether a cell is splitted):

```
Private Function isSplit(ByVal g As EXTREELibCtl.Tree, ByVal h As EXTREELibCtl.HITEM,
ByVal c As Long) As Boolean
   With g.Items
      isSplit = Iif(Not .InnerCell(h, c, 0) = .InnerCell(h, c, 1), True, False)
   End With
End Function
```

The following VB sample gets the master cell:

```
Private Function getMaster(ByVal g As EXTREELibCtl.Tree, ByVal h As EXTREELibCtl.HITEM,
ByVal c As Long) As EXTREELibCtl.HCELL
   With g.Items
      Dim r As EXTREELibCtl.HCELL
      r = c
      If Not (h = 0) Then
         r = .ItemCell(h, c)
      End If
      While Not (.CellParent(, r) = 0)
         r = .CellParent(, r)
      Wend
      getMaster = r
   End With
End Function
```

The following VB sample counts the inner cells:

```
Private Function getInnerCount(ByVal g As EXTREELibCtl.Tree, ByVal h As
EXTREELibCtl.HITEM, ByVal c As Long) As Long
   With g.Items
      Dim i As Long
      i = -1
      Do
         i = i + 1
      Loop While Not (.InnerCell(h, c, i) = .InnerCell(h, c, i + 1))
      getInnerCount = i
   End With
End Function
```

The following C++ sample specifies whether a cell contains inner cells (the function checks whether a cell is splitted):

```
long V2I( VARIANT* pvtValue )
{
   COleVariant vtResult;
   vtResult.ChangeType( VT_I4, pvtValue );
   return V_I4( &vtResult );
}
```

```
BOOL isSplit( CTree& tree, long h, long c )
{
   CItems items = tree.GetItems();
   return V2I( &items.GetInnerCell( COleVariant( h ), COleVariant( c ), COleVariant( (long)0 )
) ) != V2I( &items.GetInnerCell( COleVariant( h ), COleVariant( c ), COleVariant( (long)1 ) ) );
}
```

The following C++ sample gets the master cell:

```
long getMaster( CTree& tree, long h, long c )
{
   COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
   CItems items = tree.GetItems();
   long r = c;
   if ( h != 0 )
      r = items.GetItemCell( h, COleVariant( c ) );
   while ( V2I( &items.GetCellParent( vtMissing, COleVariant( r ) ) ) != 0 )
      r = V2I( &items.GetCellParent( vtMissing, COleVariant( r ) ) );
   return r;
}
```

The following C++ sample counts the inner cells:

```
long getInnerCount( CTree& tree, long h, long c )
{
   CItems items = tree.GetItems();
   COleVariant vtItem( h ), vtColumn( c );
   long i = -1;
   do
   {
      i++;
   }
   while ( V2I( &items.GetInnerCell( vtItem, vtColumn, COleVariant( i )  ) ) != V2I(
&items.GetInnerCell( vtItem, vtColumn, COleVariant( (long)(i + 1) )  ) ) );
   return i;
}
```

The following VB.NET sample splits the first visible cell in two cells:

```
With AxTree1.Items
    Dim i As Object
    i = .SplitCell(.FirstVisibleItem, 0)
    .CellCaption(Nothing, i) = "inner cell"
End With
```

The following C# sample splits the first visible cell in two cells:

```
EXTREELib.Items items = axTree1.Items;
object i = items.get_SplitCell(items.FirstVisibleItem, 0);
items.set_CellCaption(null, i, "inner cell");
```

The following VFP sample splits the first visible cell in two cells:

```
with thisform.Tree1.Items
    local i
    i = .SplitCell(.FirstVisibleItem,0)
    local s, crlf
    crlf = chr(13) + chr(10)
    s = "Items" + crlf
    s = s + "{" + crlf
    s = s + "CellCaption(," + str(i) + ") = " + chr(34) + "inner cell" + chr(34) + crlf
    s = s + "}"
    thisform.Tree1.Template = s
endwith
```

method Items.InsertControlItem (Parent as HITEM, ControlID as String, [License as Variant])

Inserts a new item of ActiveX type, and returns a handle to the newly created item.

Type	Description
Parent as HITEM	A long expression that indicates the handle of the parent item where the ActiveX will be inserted. If the argument is missing then the InsertControlItem property inserts the ActiveX control as a root item. If the Parent property is referring a locked item (ItemLocked property), the InsertControlItem property doesn't insert a new child ActiveX, instead insert the ActiveX control to the locked item that's specified by the Parent property.
ControlID as String	A string expression that can be formatted as follows: a prog ID, a CLSID, a URL, a reference to an Active document , a fragment of HTML.
License as Variant	A string expression that indicates the runtime license key, if it is required. An empty string, if the control doesn't require a runtime license key.

Return	Description
HITEM	A long expression that indicates the handle of the newly created item.

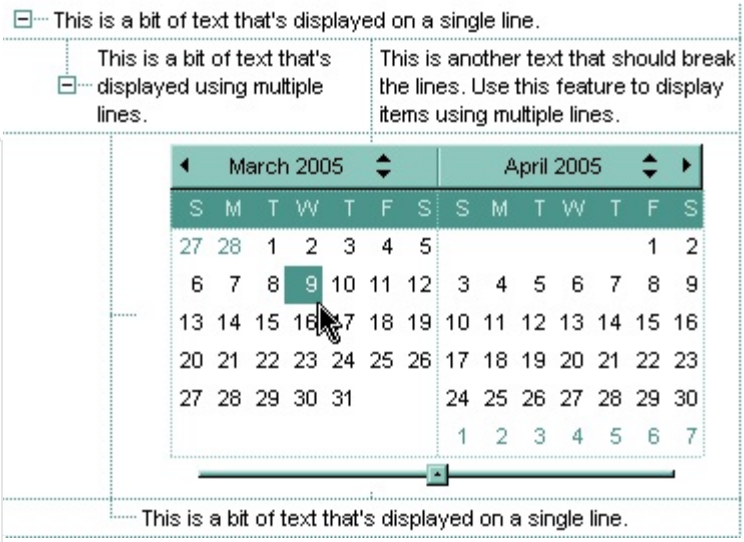
The control supports ActiveX hosting, so you can insert any ActiveX component. The ControlID must be formatted in one of the following ways:

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

The InsertControlItem property creates an ActiveX control that's hosted by the exGrid control. **The look and feel of the inner ActiveX control depends on the identifier you are using, and the version of the library that implements the ActiveX control, so you need to consult the documentation of the inner ActiveX control you are inserting inside the exTree control.**

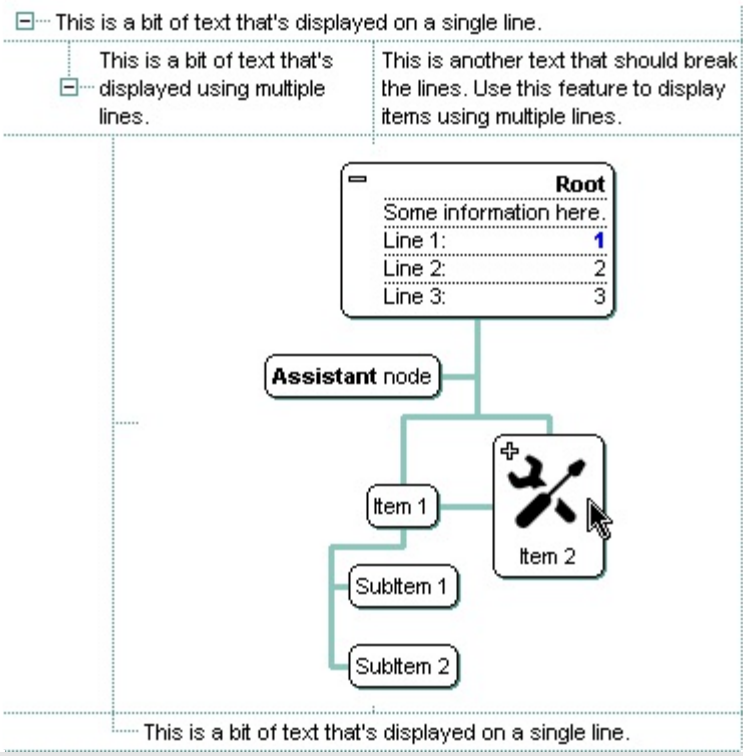
Use the ItemHeight property to specify the height of the item when it contains an ActiveX control. Use the ItemWidth property to specify the width of the ActiveX control, or the position in the item where the ActiveX is displayed. Once that an item of ActiveX type has been added you can get the OLE control created using the ItemObject property. To check if an item contains an ActiveX control you can use ItemControlID property. To change the height of an ActiveX item you have to use ItemHeight property. When the control contains at least an item of ActiveX type, it is recommended to set ScrollBySingleLine property of control to true. Events from contained components are fired through to your program using the exact same model used in VB6 for components added at run time (See ItemOleEvent event, OleEvent and OleEventParam). For instance, when an ActiveX control fires an event, the control forwards that event to your container using ItemOleEvent event of the exTree control. Use the BeginUpdate and EndUpdate methods to update the control's content when adding ActiveX controls on the fly. Use the ItemControlID property to retrieve the control's identifier.

The following VB sample adds the Exontrol's ExCalendar Component:




```
With Tree1
   .BeginUpdate
   .ScrollBySingleLine = True
   With Tree1.Items
      Dim h As HITEM
      h = .InsertControlItem(,
"Exontrol.Calendar")
      .ItemHeight(h) = 182
      With .ItemObject(h)
         .Appearance = 0
         .BackColor = vbWhite
         .ForeColor = vbBlack
         .ShowTodayButton = False
      End With
   End With
   .EndUpdate
End With
```

The following C++ sample adds the Exontrol's ExOrgChart Component:




```
#include "Items.h"

#pragma warning( disable : 4146 )
#import <ExOrgChart.dll>

CItems items = m_tree.GetItems();
m_tree.BeginUpdate();
m_tree.SetScrollBySingleLine( TRUE );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
long h = items.InsertControlItem( 0, "Exontrol.ChartView", vtMissing );
items.SetItemHeight( h, 182 );
EXORGCHARTLib::IChartViewPtr spChart( items.GetItemObject(h) );
if ( spChart != NULL )
{
   spChart->BeginUpdate();
   spChart->BackColor = RGB(255,255,255);
   spChart->ForeColor = RGB(0,0,0);
   EXORGCHARTLib::INodesPtr spNodes = spChart->Nodes;
   spNodes->Add( "Child 1", "Root", "1", vtMissing, vtMissing );
   spNodes->Add( "SubChild 1", "1", vtMissing, vtMissing, vtMissing );
   spNodes->Add( "SubChild 2", "1", vtMissing, vtMissing, vtMissing );
   spNodes->Add( "Child 2", "Root", vtMissing, vtMissing, vtMissing );
   spChart->EndUpdate();
```

```
}
m_tree.EndUpdate();
```

The sample uses the #import statement to include the ExOrgChart's Type Library. In this sample, the ItemObject property retrieves an IChartView object. The path to the library should be provided in case it is not located in your system folder.

The following C# sample adds the Exontrol's ExTree Component:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
axTree1.ScrollBySingleLine = true;
int h = items.InsertControlItem(0, "Exontrol.Tree","");
items.set_ItemHeight(h, 182);
object treeInside = items.get_ItemObject(h);
if ( treeInside != null )
{
   EXTREELib.Tree tree = treeInside as EXTREELib.Tree;
   if (tree != null)
   {
      tree.BeginUpdate();
      tree.LinesAtRoot = EXTREELib.LinesAtRootEnum.exLinesAtRoot;
      tree.Columns.Add("Column 1");
      tree.Columns.Add("Column 2");
      tree.Columns.Add("Column 3");
      EXTREELib.Items itemsInside = tree.Items;
      int hInside = itemsInside.AddItem("Item 1");
      itemsInside.set_CellCaption(hInside, 1, "SubItem 1");
      itemsInside.set_CellCaption(hInside, 2, "SubItem 2");
      hInside = itemsInside.InsertItem(hInside, null, "Item 2");
      itemsInside.set_CellCaption(hInside, 1, "SubItem 1");
      itemsInside.set_CellCaption(hInside, 2, "SubItem 2");
      tree.EndUpdate();
   }
}
axTree1.EndUpdate();
```

The following VB.NET sample adds the Exontrol's ExOrgChart Component:

```
With AxTree1
   .BeginUpdate()
   .ScrollBySingleLine = True
   With .Items
      Dim hItem As Integer
      hItem = .InsertControlItem(, "Exontrol.ChartView")
      .ItemHeight(hItem) = 182
      With .ItemObject(hItem)
         .BackColor = ToUInt32(Color.White)
         .ForeColor = ToUInt32(Color.Black)
         With .Nodes
            .Add("Child 1", , "1")
            .Add("SubChild 1", "1")
            .Add("SubChild 2", "1")
            .Add("Child 2")
         End With
      End With
   End With
   .EndUpdate()
End With
```

The following VFP sample adds the Exontrol's ExGrid Component:

```
with thisform.Tree1
   .BeginUpdate()
   .ScrollBySingleLine = .t.
   with .Items
      .DefaultItem = .InsertControlItem(0, "Exontrol.Grid")
      .ItemHeight( 0 ) = 182
      with .ItemObject( 0 )
         .BeginUpdate()
         with .Columns
            with .Add("Column 1").Editor()
               .EditType = 1 && EditType editor
            endwith
         endwith
         with .Items
```

```
            .AddItem("Text 1")
            .AddItem("Text 2")
            .AddItem("Text 3")
        endwith
        .EndUpdate()
    endwith
  endwith
  .EndUpdate()
endwith
```

The following VB sample adds dynamically an ExTree ActiveX Control and a Microsoft Calendar Control:

```
' Inserts a new ActiveX control of Exontrol.Tree type
Dim hTree As HITEM
hTree = Tree1.Items.InsertControlItem(Tree1.Items(0), "Exontrol.Tree", runtimelicensekey )
' Sets the ActiveX control height
Tree1.Items.ItemHeight(hTree) = 212
' Gets the ExTree control created. Since the ProgID used to create the item is
"Exontrol.Tree"
' the object will be of EXTREELibCtl.Tree type
Dim objTree As Object
Set objTree = Tree1.Items.ItemObject(hTree)
objTree.Columns.Add "Column"
objTree.Items.AddItem "One"
objTree.Items.AddItem "Two"
objTree.Items.AddItem "Three"

' Inserts a new ActiveX control of MSCAL.Calendar type
Dim hCalc As HITEM
hCalc = objTree.Items.InsertControlItem(, "MSCal.Calendar")
Set objCalc = Tree1.Items.ItemObject(hCalc)
objCalc.ShowTitle = False
objCalc.ShowDateSelectors = False
```

where the runtimelicensekey is the exTree's runtime license key. Please contact us to get the exTree's runtime license key. Please notice that your development license key **is not equivalent** with the generated runtime license key. **Your order number is required**, when

requesting the control's runtime license key. If you are using the DEMO version for testing purpose, you don't need a runtime license key.

The following VB sample handles any event that a contained ActiveX fires:

```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
   On Error Resume Next
   Dim i As Long
   Debug.Print "The " & Ev.Name & " was fired. "
   If Not (Ev.CountParam = 0) Then
      Debug.Print "The event has the following parameters: "
      For i = 0 To Ev.CountParam - 1
         Debug.Print " - " & Ev(i).Name & " = " & Ev(i).Value
      Next
   End If
End Sub
```

Some of ActiveX controls requires additional window styles to be added to the conatiner window. For instance, the Web Brower added by the Tree1.Items.InsertControlItem(, "https://www.exontrol.com") won't add scroll bars, so you have to do the following:

First thing is to declare the WS_HSCROLL and WS_VSCROLL constants at the top of your module:

```
Private Const WS_VSCROLL = &H200000
Private Const WS_HSCROLL = &H100000
```

Then you need to to insert a Web control use the following lines:

```
Dim hWeb As HITEM
hWeb = Tree1.Items.InsertControlItem(, "https://www.exontrol.com")
Tree1.Items.ItemHeight(hWeb) = 196
```

Next step is adding the AddItem event handler:

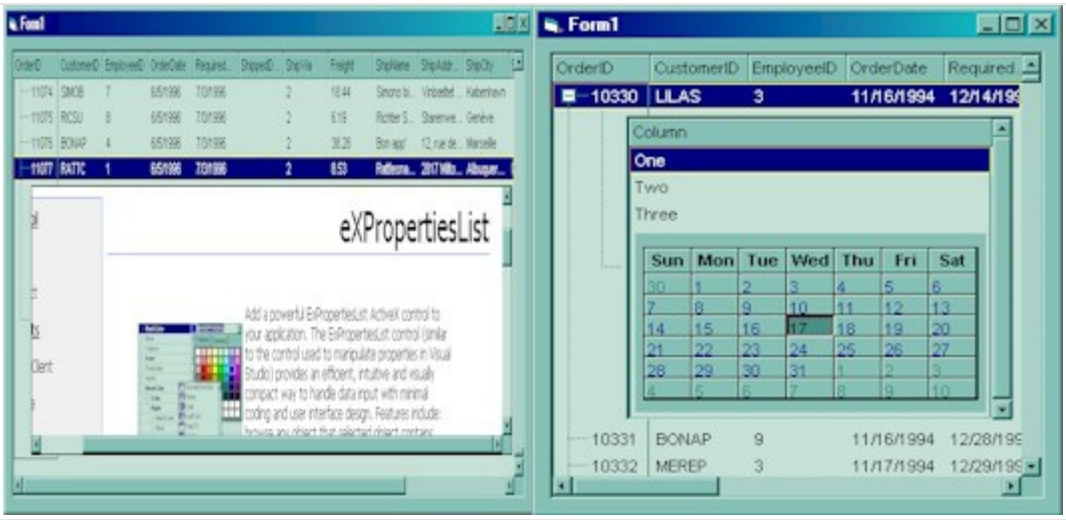
```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   If (Tree1.Items.ItemControlID(Item) = "https://www.exontrol.com") Then
      ' Some of controls like the WEB control, requires some additional window styles ( like
WS_HSCROLL and WS_VSCROLL window styles )
```

```
    ' for the window that host that WEB control, to allow scrolling the web page
    Tree1.Items.ItemWindowHostCreateStyle(Item) =
Tree1.Items.ItemWindowHostCreateStyle(Item) + WS_HSCROLL + WS_VSCROLL
  End If
End Sub
```

If somehow the InsertItemControl wasn't able to create your ActiveX on some Windows platforms, and you don't know why, you can use the following

code to make sure that ActiveX control can be created properly by using (the sample is trying to add a new Microsoft RichText ActivX control into your form):

```
Controls.Add "RICHTEXT.RichtextCtrl", "rich"
```



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.

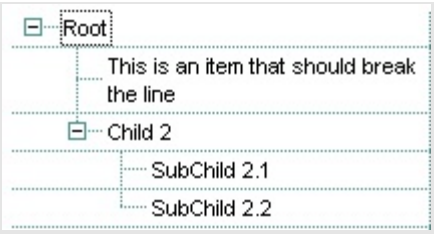
Type	Description
Parent as HITEM	A long expression that indicates the item's handle that indicates the parent item where the newly item is inserted.
UserData as Variant	A Variant expression that indicates the item's extra data.
Caption as Variant	A string expression that indicates the cell's caption on the first column, a safe array that holds the caption for each column.

Return	Description
HITEM	Retrieves the handle of the newly created item.

Use the InsertItem property to add a new child items to the specified item. The InsertItem property fires the AddItem event. You can use the InsertItem(,,"Root") or AddItem("Root") to add a root item. An item that has no parent is a root item. Use the CellCaption property to specify the cell's caption when control contains multiple columns. Use the CellCaptionFormat property to specify whether the cell displays the caption using the HTML format. To insert an ActiveX control, use the InsertControlItem property of the Items property. Use the ExpandItem property to expand an item. Use the MergeCells method to combine two or multiple cells in a single cell. Use the SplitCell property to split a cell. Use the LinesAtRoot property to link items at the root of the hierarchy. 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.

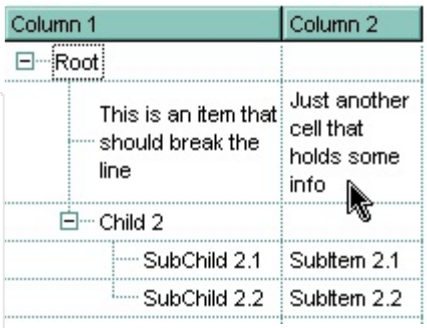
The following VB sample shows how to create a simple hierarchy:

```
With Tree1
  .BeginUpdate
  .ColumnAutoResize = True
  .LinesAtRoot = exLinesAtRoot
  .FullRowSelect = False
  .MarkSearchColumn = False
  .Columns.Add "Default"
  With .Items
    Dim h As HITEM, hx As HITEM
    h = .InsertItem(, , "Root")
```



```
      hx = .InsertItem(h, , "This is an item that should break the line")
      .CellSingleLine(hx, 0) = False
      h = .InsertItem(h, , "Child 2")
      .InsertItem h, , "SubChild 2.1"
      h = .InsertItem(h, , "SubChild 2.2")
   End With
   .EndUpdate
End With
```

The following VB sample insert items and multiple columns as well:




```
With Tree1
   .BeginUpdate
   .HeaderVisible = True
   .ColumnAutoResize = True
   .LinesAtRoot = exLinesAtRoot
   .FullRowSelect = False
   .MarkSearchColumn = False
   .Columns.Add "Column 1"
   .Columns.Add "Column 2"
   With .Items
      Dim h As HITEM, hx As HITEM
      h = .InsertItem(, , "Root")
      hx = .InsertItem(h, , Array("This is an item that should break
the line", "Just another cell that holds some info"))
      .CellSingleLine(hx, 0) = False
      .CellSingleLine(hx, 1) = False
      h = .InsertItem(h, , "Child 2")
      .InsertItem h, , Array("SubChild 2.1", "SubItem 2.1")
      h = .InsertItem(h, , Array("SubChild 2.2", "SubItem 2.2"))
   End With
   .EndUpdate
End With
```

The following VB sample inserts a child item and expands the focused item:

```
With Tree1.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_tree.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 AxTree1.Items
    Dim hItem As Integer
    hItem = .InsertItem(.FocusItem, , "new child")
    .ExpandItem(.FocusItem) = True
End With
```

The following C# sample inserts a child item and expands the focused item:

```
int hItem = axTree1.Items.InsertItem(axTree1.Items.FocusItem, null, "new child");
axTree1.Items.set_ExpandItem(axTree1.Items.FocusItem, true);
```

The following VFP sample inserts a child item and expands the focused item:

```
with thisform.Tree1.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.

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 Tree1_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 EXTREELibCtl.HitTestInfoEnum
   ' Gets the item from (X,Y)
   With Tree1
      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 OnMouseMoveTree1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
   if ( hItem != 0 )
   {
      CItems items = m_tree.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 AxTree1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseMoveEvent) Handles AxTree1.MouseMoveEvent
   With AxTree1
      Dim i As Integer, c As Integer, hit As EXTREELib.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 axTree1_MouseMoveEvent(object sender,
AxEXTREELib._ITreeEvents_MouseMoveEvent e)
{
```

```
    int c = 0;
    EXTREELib.HitTestInfoEnum hit;
    int i = axTree1.get_ItemFromPoint(e.x, e.y, out c, out hit);
    if (i != 0)
        if ( axTree1.Items.get_IsItemLocked( i ) )
        {
            object cap = axTree1.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.Tree1
    .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
    endwith
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 handle of the item that fits the client area.
Boolean	A boolean expression that indicates whether the item fits the client area.

To make sure that an item fits the client area call 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.

The following VB sample enumerates the items that fit the control's client area:

```
On Error Resume Next
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = Tree1.Columns.Count
With Tree1.Items
   h = .FirstVisibleItem
   While Not (h = 0) And .IsItemVisible(h)
      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 the items that fit the control's client area:

```
#include "Items.h"
CItems items = m_tree.GetItems();
long hItem = items.GetFirstVisibleItem();
```

```
while ( hItem && items.GetIsItemVisible( hItem ) )
{
   OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
   hItem = items.GetNextVisibleItem( hItem );
}
```

The following VB.NET sample enumerates the items that fit the control's client area:

```
With AxTree1.Items
   Dim hItem As Integer
   hItem = .FirstVisibleItem
   While Not (hItem = 0)
      If (.IsItemVisible(hItem)) Then
         Debug.Print(.CellCaption(hItem, 0))
         hItem = .NextVisibleItem(hItem)
      Else
         Exit While
      End If
   End While
End With
```

The following C# sample enumerates the items that fit the control's client area:

```
EXTREELib.Items items = axTree1.Items;
int hItem = items.FirstVisibleItem;
while ( ( hItem != 0 ) && (items.get_IsItemVisible(hItem)) )
{
   object strCaption = items.get_CellCaption(hItem, 0);
   System.Diagnostics.Debug.WriteLine( strCaption != null ? strCaption.ToString() : "" );
   hItem = items.get_NextVisibleItem(hItem);
}
```

The following VFP sample enumerates the items that fit the control's client area:

```
with thisform.Tree1.Items
   .DefaultItem = .FirstVisibleItem
   do while ( ( .DefaultItem <> 0 ) and ( .IsItemVisible( 0 ) ) )
      wait window .CellCaption( 0, 0 )
```

```
      .DefaultItem = .NextVisibleItem( 0 )
   enddo
endwith
```

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.ItemAppearance(Item as HITEM) as AppearanceEnum

Specifies the item's appearance when the item hosts an ActiveX control.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item that was previously created by InsertControlItem property.
AppearanceEnum	An AppearanceEnum expression that indicates the item's appearance.

Use the ItemAppearance property to specify the item's appearance if the item is of ActiveX type. Use the InsertControlItem property to insert an ActiveX control inside. Use the ItemObject property to access the object being created by the InsertControlItem property. Use the ItemHeight property to specify the height of the item when containing an ActiveX control.

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 handle of the item. 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. Use the SelBackColor property to change appearance for the selected items. The HTML colors are not applied if the item is selected. Use the SelectedItem property to specify whether an item is selected or unselected. Use the Add method to add new skins to the control. You can define new skins and to use it to mark some items, like in the following samples.

The following VB sample changes the item's appearance. The sample uses the "  ".

```
With Tree1
  With .VisualAppearance
    .Add &H50, App.Path + "\item.ebn"
  End With
  With .Items
    .ItemBackColor(.FirstVisibleItem) = &H50000000
  End With
End With
```

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

```
#include "Appearance.h"
#include "Items.h"
m_tree.GetVisualAppearance().Add( 0x50,
COleVariant(_T("D:\\Temp\\ExTree.Help\\item.ebn")) );
m_tree.GetItems().SetItemBackColor( m_tree.GetItems().GetFirstVisibleItem() , 0x50000000
);
```

The following VB.NET sample changes the item's appearance:

```
With AxTree1
   With .VisualAppearance
      .Add(&H50, "D:\Temp\ExTree.Help\item.ebn")
   End With
   With .Items
      .ItemBackColor(.FirstVisibleItem) = &H50000000
   End With
End With
```

The following C# sample changes the item's appearance:

```
axTree1.VisualAppearance.Add(0x50, "D:\\Temp\\ExTree.Help\\item.ebn");
axTree1.Items.set_ItemBackColor(axTree1.Items.FirstVisibleItem, 0x50000000);
```

The following VFP sample changes the item's appearance:

```
With thisform.Tree1
   With .VisualAppearance
      .Add(80, "D:\Temp\ExTree.Help\item.ebn")
   EndWith
   with .Items
      .DefaultItem = .FirstVisibleItem
      .ItemBackColor(0) = 1342177280
   endwith
EndWith
```

where the 1342177280 value represents the 0x50000000 hexa value.

The following C# sample changes the background color for the focused item:

```
axTree1.Items.set_ItemBackColor(axTree1.Items.FocusItem, ToUInt32(Color.Red) );
```

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 VB.NET sample changes the background color for the focused item:

```
With AxTree1.Items
   .ItemBackColor(.FocusItem) = ToUInt32(Color.Red)
End With
```

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 C++ sample changes the background color for the focused item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetItemBackColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the background color for the focused item:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .ItemBackColor( 0 ) = RGB(255,0,0)
```

```
endwith
```

Use the following VB sample changes the background color for the cells in the first column, when adding new items:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
    Tree1.Items.CellBackColor(Item, 0) = vbBlue
End Sub
```

property Items.ItemBold(Item as HITEM) as Boolean

Retrieves or sets a value that indicates whether the item should appear in bold.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Boolean	A boolean expression that indicates whether the item should appear in bold.

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 Tree1_SelectionChanged()
   If Not (hOldBold = 0) Then
      Tree1.Items.ItemBold(hOldBold) = False
   End If
   hOldBold = Tree1.Items.SelectedItem()
   Tree1.Items.ItemBold(hOldBold) = True
End Sub
```

The following VB sample bolds the focused item:

```
With Tree1.Items
   .ItemBold(.FocusItem) = True
End With
```

The following C++ sample bolds the focused item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetItemBold( items.GetFocusItem() , TRUE );
```

The following C# sample bolds the focused item:

```
axTree1.Items.set_ItemBold(axTree1.Items.FocusItem, true);
```

The following VB.NET sample bolds the focused item:

```
With AxTree1.Items
   .ItemBold(.FocusItem) = True
End With
```

The following VFP sample bolds the focused item:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .ItemBold( 0 ) = .t.
endwith
```

property Items.ItemByIndex (Index as Long) as HITEM

Retrieves the handle of the item given its index in Items collection..

Type	Description
Index as Long	A long expression that indicates the index of the item.
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. the 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: Tree1.Items(0), Tree1.Items.ItemByIndex(0).

The following VB sample enumerates all items in the control:

```
Dim i As Long, n As Long
With Tree1.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_tree.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 AxTree1
```

```
    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:

```
EXTREELib.Items items = axTree1.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.Tree1.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 based on a specific column.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index or the cell's handle, a string expression that indicates the column's caption.
HCELL	A long expression that indicates the handle of the cell.

A cell is the intersection of an item with a column. All properties that has an Item and a ColIndex parameters are referring to a cell. The Item parameter represents the handle of an item, and the ColIndex parameter indicates an index (a numerical value, see Column.Index property) of a column , the column's caption (a string value, see Column.Caption property), or a handle to a cell. Here's few hints how to use properties with Item and ColIndex parameters:

```
Tree1.Items.CellBold(, Tree1.Items.ItemCell(Tree1.Items(0), 0)) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), 0) = True
```

```
Tree1.Items.CellBold(Tree1.Items(0), "ColumnName") = True
```

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 this property to get the first child of an item. NextVisibleItem or NextSiblingItem to get the next visible, sibling 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 RecItem(ByVal c As EXTREELibCtl.Tree, 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)
            RecItem 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 RecItem( CTree* pTree, long hItem )
{
   COleVariant vtColumn( (long)0 );
   if ( hItem )
   {
      CItems items = pTree->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 )
        {
            RecItem( pTree, hChild );
            hChild = items.GetNextSiblingItem( hChild );
        }
    }
}
```

The following VB.NET function recursively enumerates the item and all its child items:

```
Shared Sub RecItem(ByVal c As AxEXTREELib.AxTree, 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)
                RecItem(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 RecItem(AxEXTREELib.AxTree  tree, int hItem)
{
    if (hItem != 0)
    {
        EXTREELib.Items items = tree.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(tree, 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.Tree1
    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.ItemControlID (Item as HITEM) as String

Retrieves the item's control identifier that was used by InsertControlItem property.

Type	Description
Item as HITEM	A long expression that indicates the item's handle that was previously created by the InsertControlItem property.
String	A string expression that indicates the control identifier used by InsertControlItem method to create an item that hosts an ActiveX control.

The ItemControlID property retrieves the control identifier used by the InsertControlItem property. If the item was created using AddItem or InsertItem properties the ItemControlID property retrieves an empty string. For instance, the ItemControlID property can be used to check if an item contains an ActiveX control or not.

property Items.ItemCount as Long

Retrieves the number of items.

Type	Description
Long	A long value that indicates the number of items into the Items collection.

The ItemCount property counts the items in the control. Use the ItemByIndex property to access an item giving its index. Use the VisibleItemCount property to specify the number of visible items in the list. 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, InsertControlItem, 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 Tree1.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_tree.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 AxTree1
    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:

```
EXTREELib.Items items = axTree1.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.Tree1.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 item's handle that has associated some extra data.
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 item. The value indicates the index of column used to define the divider's title.

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.

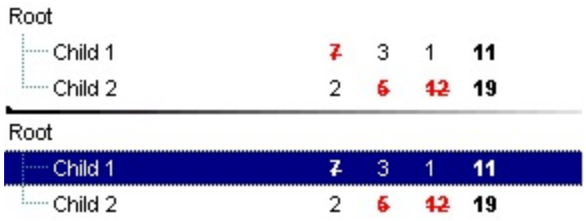
The following Template draws a custom line (from a skin object) between two items:

```
BeginUpdate

VisualAppearance.Add(1,
"gBFLBCJwBAEHhEJAEGg4BawCg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAIIRZGM



BackColor = RGB(255,255,255)
MarkSearchColumn = 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
   ExpandItem(h) = True

Dim k
k = AddItem("")
ItemDivider(k) = 0
ItemDividerLine(k) = 0
ItemHeight(k) = 4
SelectableItem(k) = False
ItemBackColor(k) = 16777216

   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
```

```
    ExpandItem(h) = True

}
EndUpdate
```



The sample loads the skin file using the Add method, and the ItemBackColor(k) = 16777216, specifies that skin with the skin with the identifier 1 (the 16777216 is the 0x1000000 in hexa representation) paints the item's background.

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 Tree1
    .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) = "<b>Total</b>:
$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"
```

```
CItems items = m_tree.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 = axTree1.Items.AddItem("divider item");
axTree1.Items.set_ItemDivider(i, 0);
axTree1.Items.set_SelectableItem(i, false);
```

The following VB.NET sample adds a divider item, that's not selectable too:

```
With AxTree1.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.Tree1.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.

By default, the ItemDividerLine property is SingleLine. The ItemDividerLine property specifies the type of line that underlines a 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 the divider item. Use the CellMerge property to merge two or more cells. Use the ItemDivider property to draw a divider item.

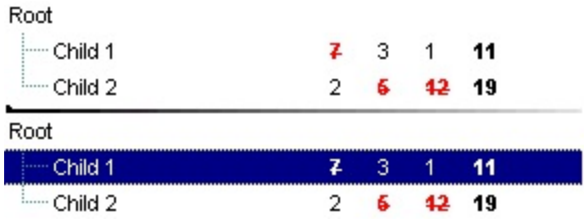
The following Template draws a custom line (from a skin object) between two items:

```
BeginUpdate

VisualAppearance.Add(1,
"gBFLBCJwBAEHhEJAEGg4BawCg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAIIRZGM



BackColor = RGB(255,255,255)
MarkSearchColumn = 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
   ExpandItem(h) = True

Dim k
k = AddItem("")
ItemDivider(k) = 0
ItemDividerLine(k) = 0
ItemHeight(k) = 4
SelectableItem(k) = False
ItemBackColor(k) = 16777216

   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
   ExpandItem(h) = True
```

```
}
EndUpdate
```



The sample loads the skin file using the Add method, and the ItemBackColor(k) = 16777216, specifies that skin with the skin with the identifier 1 (the 16777216 is the 0x1000000 in hexa representation) paints the item's background.

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.

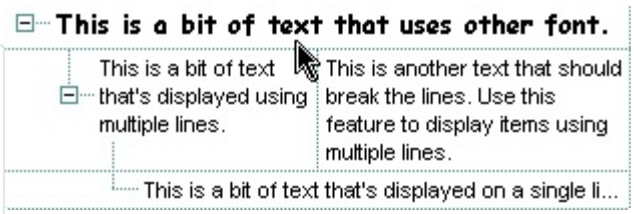
property Items.ItemFont (Item as HITEM) as IFontDisp

Retrieves or sets the item's font.

Type	Description
Item as HITEM	A long expression that specifies the item's handle.
IFontDisp	A Font object that specifies the item's font.

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 Refresh method to refresh the control's content on the fly. Use the BeginUpdate and EndUpdate methods if you are doing multiple changes, so no need for an update each time a change is done. 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 font for the focused item:




```
With Tree1.Items
   .ItemFont(.FocusItem) = Tree1.Font
   With .ItemFont(.FocusItem)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
Tree1.Refresh
```

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

```
#include "Items.h"
#include "Font.h"
CItems items = m_tree.GetItems();
items.SetItemFont( items.GetFocusItem(), m_tree.GetFont().m_lpDispatch );
COleFont font = items.GetItemFont( items.GetFocusItem() );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_tree.Refresh();
```

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

```
With AxTree1.Items
   .ItemFont(.FocusItem) = IFDH.GetIFontDisp(AxTree1.Font)
   With .ItemFont(.FocusItem)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
AxTree1.CtlRefresh()
```

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:

```
axTree1.Items.set_ItemFont( axTree1.Items.FocusItem, IFDH.GetIFontDisp( axTree1.Font ) );
stdole.IFontDisp spFont = axTree1.Items.get_ItemFont(axTree1.Items.FocusItem );
spFont.Name = "Comic Sans MS";
spFont.Bold = true;
axTree1.CtlRefresh();
```

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:

```
with thisform.Tree1.Items
    .DefaultItem = .FocusItem
    .ItemFont(0) = thisform.Tree1.Font
    with .ItemFont(0)
        .Name = "Comic Sans MS"
        .Bold = .t.
    endwith
endwith
thisform.Tree1.Object.Refresh()
```

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 HTML colors are not applied if the item is selected. Use the SelectedItem property to specify whether an item is selected or unselected.

The following VB sample changes the foreground color for cells in the first column as user add new items:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Tree1.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:

```
axTree1.Items.set_ItemForeColor(axTree1.Items.FocusItem, ToUInt32(Color.Red) );
```

The following VB.NET sample changes the foreground color of the focused item:

```
With AxTree1.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_tree.GetItems();
items.SetItemForeColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the foreground color of the focused item:

```
with thisform.Tree1.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 item's handle.
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 ItemHasChidren property is False. Use the ItemHasChildren property to build a virtual tree. 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 Tree1_BeforeExpandItem(ByVal Item As EXTREELibCtl.HITEM, Cancel As Variant)
  With Tree1.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 AxTree1_BeforeExpandItem(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_BeforeExpandItemEvent) Handles AxTree1.BeforeExpandItem
  With AxTree1.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 axTree1_BeforeExpandItem(object sender,
AxEXTREELib._ITreeEvents_BeforeExpandItemEvent e)
{
   EXTREELib.Items items = axTree1.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 OnBeforeExpandItemTree1(long Item, VARIANT FAR* Cancel)
{
   CItems items = m_tree.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.Tree1.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 item's handle. 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 in pixels.

To change the default height of the item before inserting items to collection you can call DefaultItemHeight property of the control. The control supports items with different heights. When an item hosts an ActiveX control (was previously created by the InsertControlItem property), the ItemHeight property changes the height of contained ActiveX control. 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 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 that uses italic font attribute.
Boolean	A boolean expression that indicates whether the item should appear in italic.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the 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 Tree1_SelectionChanged()
   If Not (h = 0) Then Tree1.Items.ItemItalic(h) = False
   h = Tree1.Items.SelectedItem()
   Tree1.Items.ItemItalic(h) = True
End Sub
```

The following VB sample makes italic the focused item:

```
With Tree1.Items
   .ItemItalic(.FocusItem) = True
End With
```

The following C++ sample makes italic the focused item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetItemItalic( items.GetFocusItem() , TRUE );
```

The following C# sample makes italic the focused item:

```
axTree1.Items.set_ItemItalic(axTree1.Items.FocusItem, true);
```

The following VB.NET sample makes italic the focused item:

```
With AxTree1.Items
   .ItemItalic(.FocusItem) = True
End With
```

The following VFP sample makes italic the focused item:

```
with thisform.Tree1.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.

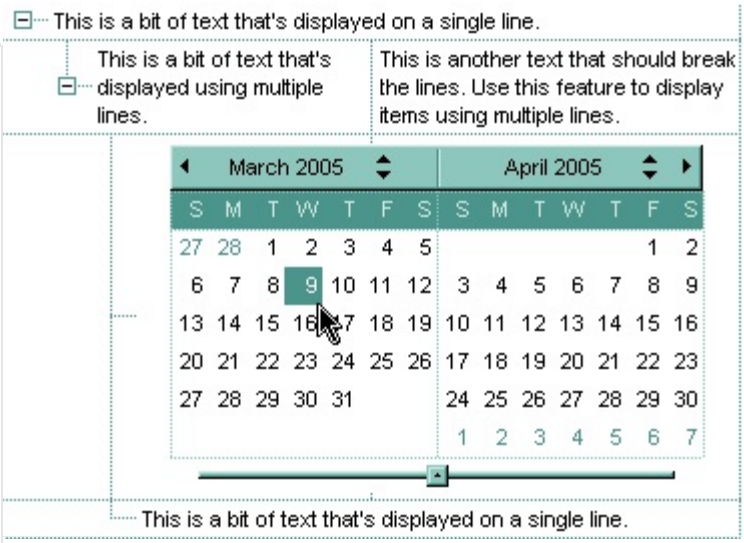
property Items.ItemObject (Item as HITEM) as Object

Retrieves the item's ActiveX object associated, if the item was previously created by InsertControlItem property. /*not supported in the lite version*/

Type	Description
Item as HITEM	A long expression that indicates the handle of the item that was previously created by InsertControlItem property.
Object	An object that indicates the ActiveX hosted by the item.

Use the ItemObject to retrieve the ActiveX control created by the InsertControlItem method. Use the ItemControlID property to retrieve the control's identifier. Use the ItemHeight property to specify the item's height. If the item hosts an ActiveX control, the ItemHeight property specifies the height of the ActiveX control also.

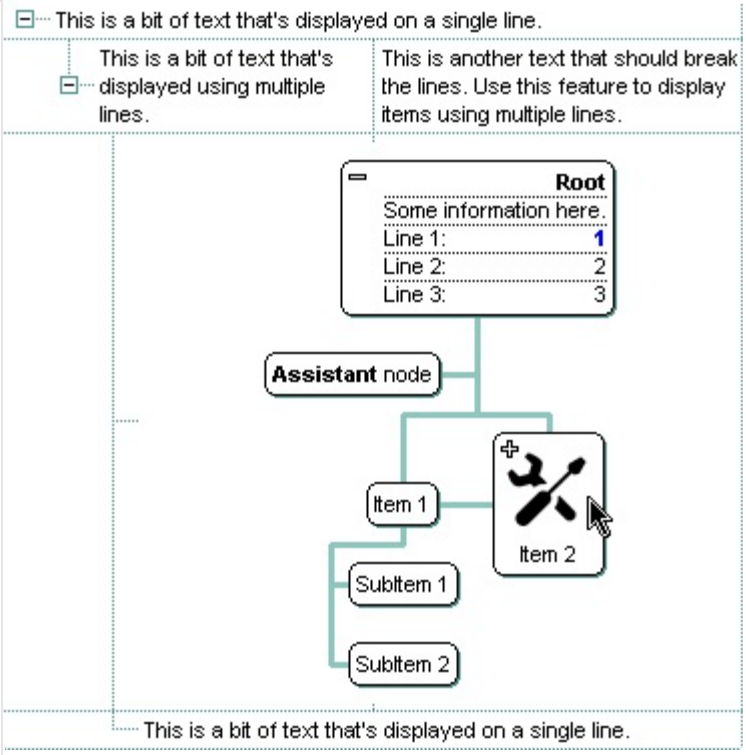
The following VB sample adds the Exontrol's ExCalendar Component:




```
With Tree1
   .BeginUpdate
   .ScrollBySingleLine = True
   With Tree1.Items
      Dim h As HITEM
      h = .InsertControlItem(,
"Exontrol.Calendar")
      .ItemHeight(h) = 182
      With .ItemObject(h)
         .Appearance = 0
         .BackColor = vbWhite
         .ForeColor = vbBlack
         .ShowTodayButton = False
      End With
   End With
   .EndUpdate
End With
```

The following C++ sample adds the Exontrol's ExOrgChart Component:

```
#include "Items.h"

#pragma warning( disable : 4146 )
#import <ExOrgChart.dll>

CItems items = m_tree.GetItems();
m_tree.BeginUpdate();
m_tree.SetScrollBySingleLine( TRUE );
COleVariant vtMissing; V_VT( &vtMissing ) =
VT_ERROR;
long h = items.InsertControlItem( 0,
"Exontrol.ChartView", vtMissing );
items.SetItemHeight( h, 182 );
EXORGCHARTLib::IChartViewPtr spChart(
items.GetItemObject(h) );
if ( spChart != NULL )
{
   spChart->BeginUpdate();
   spChart->BackColor = RGB(255,255,255);
   spChart->ForeColor = RGB(0,0,0);
   EXORGCHARTLib::INodesPtr spNodes =
spChart->Nodes;
   spNodes->Add( "Child 1", "Root", "1",
vtMissing, vtMissing );
   spNodes->Add( "SubChild 1", "1", vtMissing,
vtMissing, vtMissing );
   spNodes->Add( "SubChild 2", "1", vtMissing,
vtMissing, vtMissing );
   spNodes->Add( "Child 2", "Root", vtMissing,
vtMissing, vtMissing );
   spChart->EndUpdate();
}
m_tree.EndUpdate();
```



The sample uses the #import statement to include the ExOrgChart's Type Library. In this sample, the ItemObject property retrieves an IChartView object. The path to the library should be provided in case it is not located in your system folder.

The following C# sample adds the Exontrol's ExTree Component:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
axTree1.ScrollBySingleLine = true;
int h = items.InsertControlItem(0, "Exontrol.Tree","");
items.set_ItemHeight(h, 182);
object treeInside = items.get_ItemObject(h);
if ( treeInside != null )
{
   EXTREELib.Tree tree = treeInside as EXTREELib.Tree;
   if (tree != null)
   {
      tree.BeginUpdate();
      tree.LinesAtRoot = EXTREELib.LinesAtRootEnum.exLinesAtRoot;
      tree.Columns.Add("Column 1");
      tree.Columns.Add("Column 2");
      tree.Columns.Add("Column 3");
      EXTREELib.Items itemsInside = tree.Items;
      int hInside = itemsInside.AddItem("Item 1");
      itemsInside.set_CellCaption(hInside, 1, "SubItem 1");
      itemsInside.set_CellCaption(hInside, 2, "SubItem 2");
      hInside = itemsInside.InsertItem(hInside, null, "Item 2");
      itemsInside.set_CellCaption(hInside, 1, "SubItem 1");
      itemsInside.set_CellCaption(hInside, 2, "SubItem 2");
      tree.EndUpdate();
   }
}
axTree1.EndUpdate();
```

The following VB.NET sample adds the Exontrol's ExOrgChart Component:

```
With AxTree1
   .BeginUpdate()
   .ScrollBySingleLine = True
   With .Items
      Dim hItem As Integer
      hItem = .InsertControlItem(, "Exontrol.ChartView")
```

```
    .ItemHeight(hItem) = 182
    With .ItemObject(hItem)
      .BackColor = ToUInt32(Color.White)
      .ForeColor = ToUInt32(Color.Black)
      With .Nodes
        .Add("Child 1", , "1")
        .Add("SubChild 1", "1")
        .Add("SubChild 2", "1")
        .Add("Child 2")
      End With
    End With
  End With
  .EndUpdate()
End With
```

The following VFP sample adds the Exontrol's ExGrid Component:

```
with thisform.Tree1
  .BeginUpdate()
  .ScrollBySingleLine = .t.
  with .Items
    .DefaultItem = .InsertControlItem(0, "Exontrol.Grid")
    .ItemHeight( 0 ) = 182
    with .ItemObject( 0 )
      .BeginUpdate()
      with .Columns
        with .Add("Column 1").Editor()
          .EditType = 1 && EditType editor
        endwith
      endwith
      with .Items
        .AddItem("Text 1")
        .AddItem("Text 2")
        .AddItem("Text 3")
      endwith
      .EndUpdate()
    endwith
```

```
    endwith
    .EndUpdate()
endwith
```

property Items.ItemParent (Item as HITEM) as HITEM

Returns the handle of the parent 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 parent item.

Use the ItemParent property to retrieve the parent item. Use the InsertItem property to insert child items. Use the InsertControlItem property to insert ActiveX controls. The SetParent method changes the item's parent at runtime. 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. To get the collection of root items you can use RootCount and RootItem properties. Use the ItemChild property to retrieve the first child 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 item's handle.
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. You can use the ItemPosition property to change the item's position after it been added to collection. When the control sorts the tree, the item for each position can be changed, so you can use the item's handle or item's index to identify an item. 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 a value that indicates whether the item should appear in strikeout.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item should appear in strikeout.

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 Tree1_SelectionChanged()
   If Not (h = 0) Then Tree1.Items.ItemStrikeOut(h) = False
   h = Tree1.Items.SelectedItem()
   Tree1.Items.ItemStrikeOut(h) = True
End Sub
```

The following VB sample draws a horizontal line through the focused item:

```
With Tree1.Items
   .ItemStrikeOut(.FocusItem) = True
End With
```

The following C++ sample draws a horizontal line through the focused item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetItemStrikeOut( items.GetFocusItem() , TRUE );
```

The following C# sample draws a horizontal line through the focused item:

```
axTree1.Items.set_ItemStrikeOut(axTree1.Items.FocusItem, true);
```

The following VB.NET sample draws a horizontal line through the focused item:

```
With AxTree1.Items
   .ItemStrikeOut(.FocusItem) = True
End With
```

The following VFP sample draws a horizontal line through the focused item:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .ItemStrikeOut( 0 ) = .t.
endwith
```

property Items.ItemToIndex (Item as HITEM) as Long

Retrieves the index of item into 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 a value that indicates whether the item should appear in underline.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item should appear in underline.

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 Tree1_SelectionChanged()
   If Not (h = 0) Then Tree1.Items.ItemUnderline(h) = False
   h = Tree1.Items.SelectedItem()
   Tree1.Items.ItemUnderline(h) = True
End Sub
```

The following VB sample underlines the focused item:

```
With Tree1.Items
   .ItemUnderline(.FocusItem) = True
End With
```

The following C++ sample underlines the focused item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetItemUnderline( items.GetFocusItem() , TRUE );
```

The following C# sample underlines the focused item:

```
axTree1.Items.set_ItemUnderline(axTree1.Items.FocusItem, true);
```

The following VB.NET sample underlines the focused item:

```
With AxTree1.Items
   .ItemUnderline(.FocusItem) = True
End With
```

The following VFP sample underlines the focused item:

```
with thisform.Tree1.Items
   .DefaultItem = .FocusItem
   .ItemUnderline( 0 ) = .t.
endwith
```

property Items.ItemWidth(Item as HITEM) as Long

Retrieves or sets a value that indicates the item's width while it contains an ActiveX control.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long expression that indicates the item's width, when the item contains an ActiveX control.

By default, the ItemWidth property is -1. If the ItemWidth property is -1, the control resizes the ActiveX control to fit the control's client area. Use the ItemHeight property to specify the item's height. The property has effect only if the item contains an ActiveX control. Use the InsertControlItem property to insert ActiveX controls. Use the ItemObject property to retrieve the ActiveX object that's hosted by an item.

The ItemWidth property is interpreted like follows:

- If the ItemWidth property is greater than zero, the ItemWidth property indicates the width in pixels of the ActiveX control. The TreeColumnIndex property indicates the column where the ActiveX control is shown. For instance, ItemWidth = 64, indicates that the width of the inside ActiveX control is 64 pixels.
- If the ItemWidth property is zero, the ActiveX control uses the full item area to display the inside ActiveX control.
- If the ItemWidth property is -1, the TreeColumnIndex property indicates the column where the ActiveX control is shown and the inside ActiveX control is shown to the end of the control.
- If the ItemWidth property is less than -32000, the formula -(ItemWidth+32000) indicates the index of the column where the inside ActiveX is displayed. For instance, -32000 indicates that the cell in the first column displays the inside ActiveX control, -32001 indicates that the cell in the second column displays the inside ActiveX control, -32002 indicates that the cell in the third column displays the inside ActiveX control, and so on.
- If the ItemWidth property is -InnerCell or ItemCell, the ItemWidth property indicates the handle of the cell that shows the inside ActiveX. This option should be used when you need to display the ActiveX control in an inner cell. Use the SplitCell property to create inner cells, to divide a cell or to split a cell. For instance, .ItemWidth(.FirstVisibleItem) = -.InnerCell(.FirstVisibleItem, 1, 1) indicates that the inside ActiveX control is shown in the second inner cell in the second column, in the first visible item. Use the CellWidth property to specify the width of the inner cell.

property Items.ItemWindowHost (Item as HITEM) as Long

Retrieves the window's handle that hosts an ActiveX control when the item was created using InsertControlItem method.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item that was previously created by InsertControlItem method.
Long	A long value that indicates the window handle that hosts the item's ActiveX.

The ItemWindowHost property retrieves the handle of the window that's the container for the item's ActiveX control. Use the InserControlItem method to insert an ActiveX control. Use the ItemObject property to access the ActiveX properties and methods. Use the hWnd property to get the handle of the control's window. The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

property Items.ItemWindowHostCreateStyle(Item as HITEM) as Long

Retrieves or sets a value that indicates a combination of window styles used to create the ActiveX window host. /*not supported in the lite version*/

Type	Description
Item as HITEM	A long expression that indicates the handle of the item that was previously created by InsertControlItem method.
Long	A long value that indicates the container window's style.

The ItemWindowHostCreateStyle property specifies the window styles of the ActiveX's container window, when a new ActiveX control is inserted using the InsertControlItem method. The ItemWindowHostCreateStyle property has no effect for non ActiveX items. The ItemWindowHostCreateStyle property must be called during the AddItem event, like in the following samples. Generally, the ItemWindowHostCreateStyle property is useful to include WS_HSCROLL and WS_VSCROLL styles for a IWebBrowser control (WWW browser control), to include scrollbars in the browsed web page.

Some of ActiveX controls requires additional window styles to be added to the container window. For instance, the Web Brower added by the Tree1.Items.InsertControlItem(, "https://www.exontrol.com") won't add scroll bars, so you have to do the following:

First thing is to declare the WS_HSCROLL and WS_VSCROLL constants at the top of your module:

```
Private Const WS_VSCROLL = &H200000
Private Const WS_HSCROLL = &H100000
```

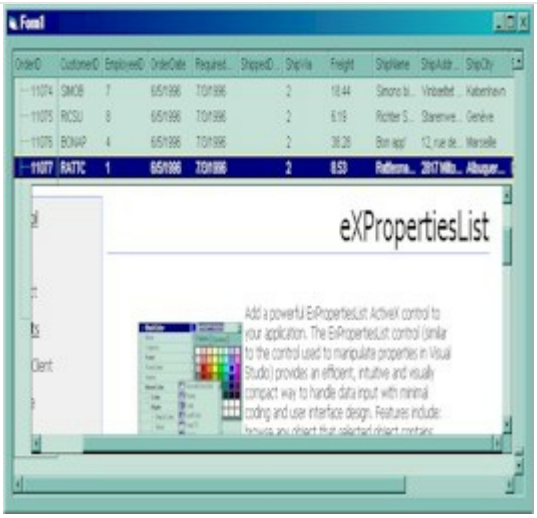
Then you need to to insert a Web control use the following lines:

```
Dim hWeb As HITEM
hWeb = Tree1.Items.InsertControlItem(, "https://www.exontrol.com")
Tree1.Items.ItemHeight(hWeb) = 196
```

Next step is adding the AddItem event handler:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
    If (Tree1.Items.ItemControlID(Item) = "https://www.exontrol.com") Then
        ' Some of controls like the WEB control, requires some additional window styles ( like WS_HSCROLL and WS_VSCROLL window styles )
        ' for the window that host that WEB control, to allow scrolling the web page
```

```
        Tree1.Items.ItemWindowHostCreateStyle(Item) =
Tree1.Items.ItemWindowHostCreateStyle(Item) + WS_HSCROLL + WS_VSCROLL
    End If
End Sub
```



property Items.LastVisibleItem ([Partially as Variant]) as HITEM

Retrieves the handle of the last visible item.

Type	Description
Partially as Variant	A Boolean expression that indicates whether the item is partially visible. By default, the Partially parameter is False.
HITEM	A long expression that indicates 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.

The following VB sample enumerates the items that fit the control's client area:

```
On Error Resume Next
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = Tree1.Columns.Count
With Tree1.Items
   h = .FirstVisibleItem
   While Not (h = 0) And .IsItemVisible(h)
      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 the items that fit the control's client area:

```
#include "Items.h"
CItems items = m_tree.GetItems();
```

```
long hItem = items.GetFirstVisibleItem();
while ( hItem && items.GetIsItemVisible( hItem ) )
{
   OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
   hItem = items.GetNextVisibleItem( hItem );
}
```

The following VB.NET sample enumerates the items that fit the control's client area:

```
With AxTree1.Items
   Dim hItem As Integer
   hItem = .FirstVisibleItem
   While Not (hItem = 0)
      If (.IsItemVisible(hItem)) Then
         Debug.Print(.CellCaption(hItem, 0))
         hItem = .NextVisibleItem(hItem)
      Else
         Exit While
      End If
   End While
End With
```

The following C# sample enumerates the items that fit the control's client area:

```
EXTREELib.Items items = axTree1.Items;
int hItem = items.FirstVisibleItem;
while ( ( hItem != 0 ) && (items.get_IsItemVisible(hItem)) )
{
   object strCaption = items.get_CellCaption(hItem, 0);
   System.Diagnostics.Debug.WriteLine( strCaption != null ? strCaption.ToString() : "" );
   hItem = items.get_NextVisibleItem(hItem);
}
```

The following VFP sample enumerates the items that fit the control's client area:

```
with thisform.Tree1.Items
   .DefaultItem = .FirstVisibleItem
   do while ( ( .DefaultItem <> 0 ) and ( .IsItemVisible( 0 ) ) )
```

```
    wait window .CellCaption( 0, 0 )
    .DefaultItem = .NextVisibleItem( 0 )
  enddo
endwith
```

property Items.LockedItem (Alignment as VAlignmentEnum, Index as Long) as HITEM

Retrieves the handle of the locked item.

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. 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 InsertControlItem property to assign an ActiveX control to a locked item only

The following VB sample adds an item that's locked to the top side of the control:

```
With Tree1
  Dim a As EXTREELibCtl.VAlignmentEnum
  a = EXTREELibCtl.VAlignmentEnum.TopAlignment
  .BeginUpdate
  With .Items
    .LockedItemCount(a) = 1
    Dim h As EXTREELibCtl.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_tree.BeginUpdate();
CItems items = m_tree.GetItems();
items.SetLockedItemCount( 0 /*TopAlignment*/, 1);
long i = items.GetLockedItem( 0 /*TopAlignment*/, 0 );
COleVariant vtItem(i), vtColumn( long(0) );
items.SetCellCaption( vtItem, vtColumn, COleVariant( "<b>locked</b> item" ) );
items.SetCellCaptionFormat( vtItem, vtColumn, 1/*exHTML*/ );
m_tree.EndUpdate();
```

The following VB.NET sample adds an item that's locked to the top side of the control:

```
With AxTree1
   .BeginUpdate()
   With .Items
      .LockedItemCount(EXTREELib.VAlignmentEnum.TopAlignment) = 1
      Dim i As Integer
      i = .LockedItem(EXTREELib.VAlignmentEnum.TopAlignment, 0)
      .CellCaption(i, 0) = "<b>locked</b> item"
      .CellCaptionFormat(i, 0) = EXTREELib.CaptionFormatEnum.exHTML
   End With
   .EndUpdate()
End With
```

The following C# sample adds an item that's locked to the top side of the control:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
items.set_LockedItemCount(EXTREELib.VAlignmentEnum.TopAlignment, 1);
int i = items.get_LockedItem(EXTREELib.VAlignmentEnum.TopAlignment, 0);
items.set_CellCaption(i, 0, "<b>locked</b> item");
items.set_CellCaptionFormat(i, 0, EXTREELib.CaptionFormatEnum.exHTML);
axTree1.EndUpdate();
```

The following VFP sample adds an item that's locked to the top side of the control:

```
with thisform.Tree1
   .BeginUpdate()
```

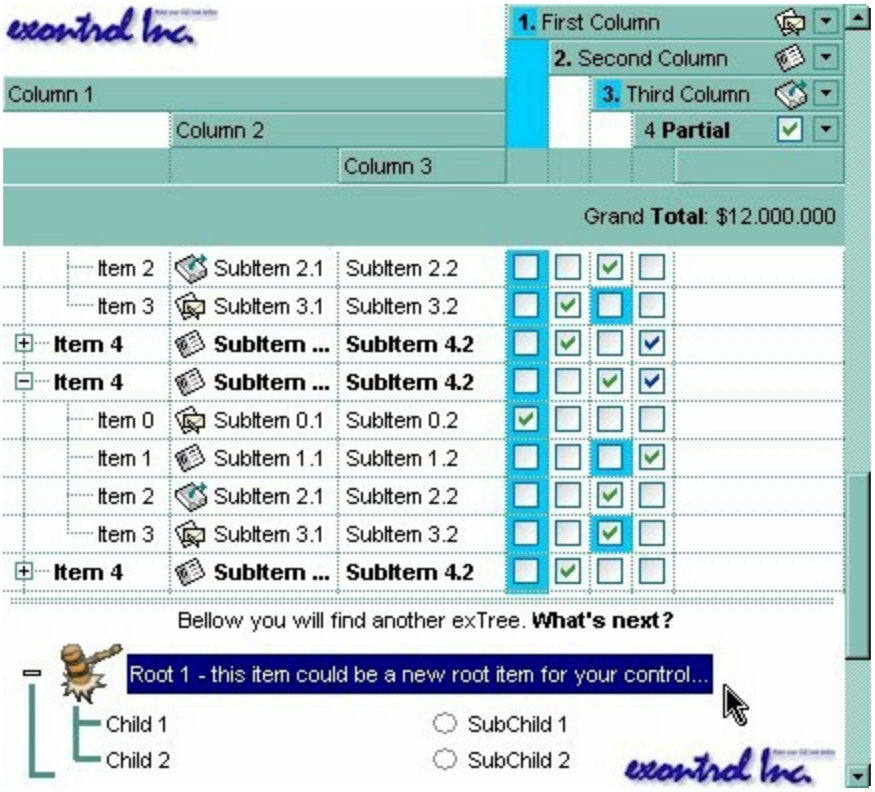
```
      With .Items
         .LockedItemCount(0) = 1
         .DefaultItem = .LockedItem(0, 0)
         .CellCaption(0, 0) = "<b>locked</b> item"
         .CellCaptionFormat(0, 0) = 1 && EXTREELib.CaptionFormatEnum.exHTML
      EndWith
   .EndUpdate()
endwith
```

property Items.LockedItemCount(Alignment as VAlignmentEnum) as Long

Specifies the number of items fixed on the top or bottom side of the control.

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. 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 multiple cells in a single cell.



The following VB sample adds two items that are locked to the top side of the control, and one item that's locked to the bottom side of the control:

```
With Tree1
```

```
  Dim h As EXTREELibCtl.HITEM
  Dim a As EXTREELibCtl.VAlignmentEnum
  a = EXTREELibCtl.VAlignmentEnum.TopAlignment
  .BeginUpdate
  With .Items
    .LockedItemCount(a) = 2
    For i = 0 To .LockedItemCount(a) - 1
      h = .LockedItem(a, i)
      .CellCaption(h, 0) = "item <b>locked</b> to the top side of the control"
      .CellCaptionFormat(h, 0) = exHTML
      .ItemBackColor(h) = SystemColorConstants.vb3DFace
      .ItemForeColor(h) = SystemColorConstants.vbWindowText
    Next
    a = EXTREELibCtl.VAlignmentEnum.BottomAlignment
    .LockedItemCount(a) = 1
    h = .LockedItem(a, 0)
    .CellCaption(h, 0) = "item <b>locked</b> to the bottom side of the control"
    .CellCaptionFormat(h, 0) = exHTML
    .ItemBackColor(h) = SystemColorConstants.vb3DFace
  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_tree.BeginUpdate();
CItems items = m_tree.GetItems();
items.SetLockedItemCount( 0 /*TopAlignment*/, 1);
long i = items.GetLockedItem( 0 /*TopAlignment*/, 0 );
COleVariant vtItem(i), vtColumn( long(0) );
items.SetCellCaption( vtItem, vtColumn, COleVariant( "<b>locked</b> item" ) );
items.SetCellCaptionFormat( vtItem, vtColumn, 1/*exHTML*/ );
m_tree.EndUpdate();
```

The following VB.NET sample adds an item that's locked to the top side of the control:

```
With AxTree1
```

```
  .BeginUpdate()
  With .Items
    .LockedItemCount(EXTREELib.VAlignmentEnum.TopAlignment) = 1
    Dim i As Integer
    i = .LockedItem(EXTREELib.VAlignmentEnum.TopAlignment, 0)
    .CellCaption(i, 0) = "<b>locked</b> item"
    .CellCaptionFormat(i, 0) = EXTREELib.CaptionFormatEnum.exHTML
  End With
  .EndUpdate()
End With
```

The following C# sample adds an item that's locked to the top side of the control:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
items.set_LockedItemCount(EXTREELib.VAlignmentEnum.TopAlignment, 1);
int i = items.get_LockedItem(EXTREELib.VAlignmentEnum.TopAlignment, 0);
items.set_CellCaption(i, 0, "<b>locked</b> item");
items.set_CellCaptionFormat(i, 0, EXTREELib.CaptionFormatEnum.exHTML);
axTree1.EndUpdate();
```

The following VFP sample adds an item that's locked to the top side of the control:

```
with thisform.Tree1
  .BeginUpdate()
    With .Items
      .LockedItemCount(0) = 1
      .DefaultItem = .LockedItem(0, 0)
      .CellCaption(0, 0) = "<b>locked</b> item"
      .CellCaptionFormat(0, 0) = 1 && EXTREELib.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:

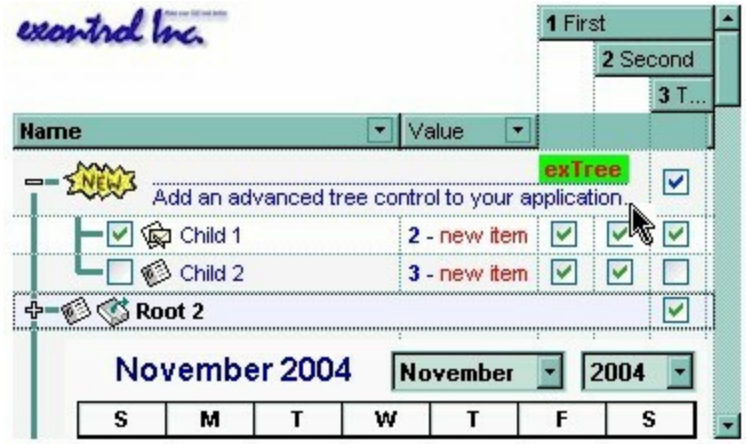
- 0, the control displays/contains no items, and no filter is applied to any column
- -1, the control displays no items, and there is a filter applied (no match found)
- positive number, indicates the number of items within the control (ItemCount property)
- negative number, the absolute value minus 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.

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 retrieves 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 retrieves 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.

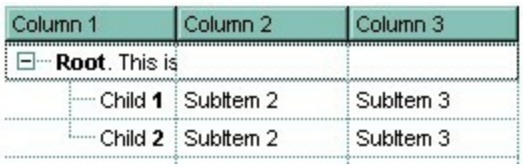
The MergeCells method combines two or more cells into one cell. The data in the **first specified cell** is displayed in the new larger cell. All the other cells' data is not lost. Use the CellMerge property to merge or unmerge a cell with another cell in the same item. Use the ItemDivider property to display a single cell in the entire item (merging all cells in the 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 retrieves 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 SplitCell property to split a cell. Use the SelectableItem property to specify the user can select an item.



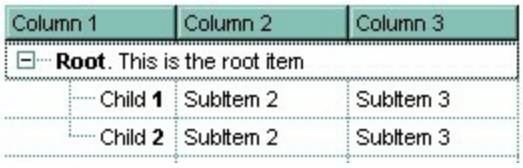
The following VB sample adds three columns, a root item and two child items:

```
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):



If we are merging the first three cells in the root item we get:



You can merge the first three cells in the root item using any of the following methods:

```
With Tree1
   With .Items
```

```
      .CellMerge(.RootItem(0), 0) = Array(1, 2)
   End With
End With
```

```
With Tree1
   .BeginUpdate
   With .Items
      Dim r As Long
      r = .RootItem(0)
      .CellMerge(r, 0) = 1
      .CellMerge(r, 0) = 2
   End With
   .EndUpdate
End With
```

```
With Tree1
   .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 Tree1
   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 Tree1
   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 Tree1.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_tree.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 AxTree1.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:

```
EXTREELib.Items items = axTree1.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.Tree1.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.

The following VB function recursively enumerates the item and all its child items:

```
Sub RecItem(ByVal c As EXTREELibCtl.Tree, 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)
            RecItem 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 RecItem( CTree* pTree, long hItem )
{
   COleVariant vtColumn( (long)0 );
   if ( hItem )
   {
      CItems items = pTree->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 )
      {
         RecItem( pTree, hChild );
         hChild = items.GetNextSiblingItem( hChild );
      }
   }
}
```

The following VB.NET function recursively enumerates the item and all its child items:

```
Shared Sub RecItem(ByVal c As AxEXTREELib.AxTree, 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)
            RecItem(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 RecItem(AxEXTREELib.AxTree  tree, int hItem)
{
   if (hItem != 0)
   {
      EXTREELib.Items items = tree.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(tree, 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.Tree1
   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 FirstVisibleItem property to retrieve the first visible item.

The following VB sample enumerates all visible items:

```
Private Sub VisItems(ByVal c As EXTREELibCtl.Tree)
   Dim h As HITEM
   With c.Items
      h = .FirstVisibleItem
      While Not (h = 0)
         Debug.Print .CellCaption(h, 0)
         h = .NextVisibleItem(h)
      Wend
   End With
End Sub
```

The following C++ sample enumerates all visible items:

```
#include "Items.h"
CItems items = m_tree.GetItems();
long hItem = items.GetFirstVisibleItem();
while ( hItem )
{
   OutputDebugString( V2S( &items.GetCellCaption( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
   hItem = items.GetNextVisibleItem( hItem );
}
```

The following C# sample enumerates all visible items:

```
EXTREELib.Items items = axTree1.Items;
int hItem = items.FirstVisibleItem;
while ( hItem != 0 )
{
   object strCaption = items.get_CellCaption(hItem, 0);
   System.Diagnostics.Debug.WriteLine( strCaption != null ? strCaption.ToString() : "" );
   hItem = items.get_NextVisibleItem(hItem);
}
```

The following VB.NET sample enumerates all visible items:

```
With AxTree1.Items
   Dim hItem As Integer
   hItem = .FirstVisibleItem
   While Not (hItem = 0)
      Debug.Print(.CellCaption(hItem, 0))
      hItem = .NextVisibleItem(hItem)
   End While
End With
```

The following VFP sample enumerates all visible items:

```
with thisform.Tree1.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

Use the RemoveAllItems method to remove all items in the control. Use the Clear method to remove all columns in the control. Use the RemoveItem method to remove a single item in the control.

method Items.RemoveItem (Item as HITEM)

Removes a specific item.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being removed.

The RemoveItem method removes an item. The RemoveItem method does not remove the item, if it contains child items. The following sample removes the first item: Tree1.Items.RemoveItem Tree1.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. The RemoveSelection method removes the selected items (including the descendents).

The following VB sample removes recursively an item:

```
Private Sub RemoveItemRec(ByVal t As EXTREELibCtl.Tree, 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( CTree* pTree, long hItem )
```

```
{
   if ( hItem )
   {
      pTree->BeginUpdate();
      CItems items = pTree->GetItems();
      long hChild = items.GetItemChild( hItem );
      while ( hChild )
      {
         long nNext = items.GetNextSiblingItem( hChild );
         RemoveItemRec( pTree, hChild );
         hChild = nNext;
      }
      items.RemoveItem( hItem );
      pTree->EndUpdate();
   }
}
```

The following VB.NET sample removes recursively an item:

```
Shared Sub RemoveItemRec(ByVal t As AxEXTREELib.AxTree, 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(AxEXTREELib.AxTree tree, int hItem)
```

```
{
   if (hItem != 0)
   {
      EXTREELib.Items items = tree.Items;
      tree.BeginUpdate();
      int hChild = items.get_ItemChild(hItem);
      while (hChild != 0)
      {
         int hNext = items.get_NextSiblingItem(hChild);
         RemoveItemRec(tree, hChild);
         hChild = hNext;
      }
      items.RemoveItem(hItem);
      tree.EndUpdate();
   }
}
```

The following VFP sample removes recursively an item (removeitemrec method):

```
LPARAMETERS h

with thisform.Tree1
   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 includes no descendents). The UnselectAll method unselects all items in the list.

property Items.RootCount as Long

Retrieves the number of root objects into Items collection.

Type	Description
Long	A long value that indicates the count of root items in the 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 Tree1.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_tree.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 AxTree1.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 < axTree1.Items.RootCount; i++)
{
   object strCaption = axTree1.Items.get_CellCaption(axTree1.Items.get_RootItem(i), 0);
   System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following VFP sample enumerates all root items:

```
with thisform.Tree1.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 giving its index into the root items collection.

Type	Description
Position as Long	A long value that indicates the position of the root item being accessed.
HITEM	A long expression that indicates the handle of 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 Tree1.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_tree.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 AxTree1.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 < axTree1.Items.RootCount; i++)
{
    object strCaption = axTree1.Items.get_CellCaption(axTree1.Items.get_RootItem(i), 0);
    System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following VFP sample enumerates all root items:

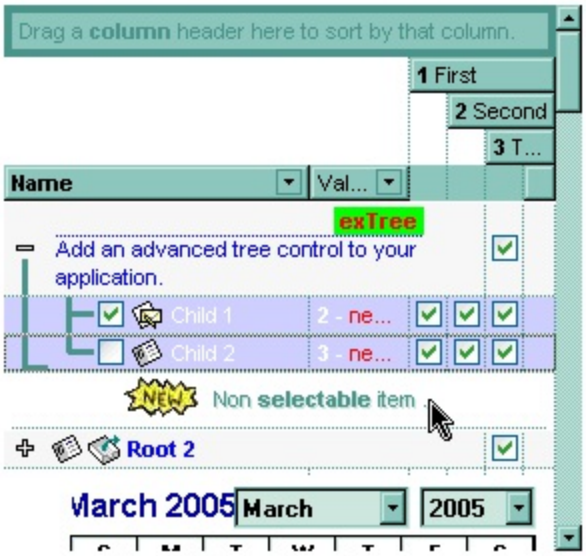
```
with thisform.Tree1.Items
    local i
    for i = 0 to .RootCount - 1
        .DefaultItem = .RootItem(i)
        wait window nowait .CellCaption(0,0)
    next
endwith
```

property Items.SelectableItem(Item as HITEM) as Boolean

Specifies whether the user can select the item.

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 ItemDivider property to define a divider item. Use the ItemForeColor property to specify the item's foreground color. Use the ItemBackColor property to specify the item's background color. Use the ItemFont, ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to assign a different font to the item. Use the EnableItem property to disable an item. A disabled item looks grayed, but it is selectable. For instance, the user can't change the check box state in a disabled item. Use the SelectItem property to select an item. The ItemFromPoint property gets the item from point. For instance, if the user clicks a non selectable item the SelectionChanged event is not fired. A non selectable item is not focusable as well. It means that if the incremental searching is on, the non selectable items are ignored. Use the SelectCount property to get the number of selected items. Use the SelForeColor and SelBackColor properties to customize the colors for selected items.



The following VB sample makes not selectable the first visible item:

```
With Tree1.Items
```

```
    .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C++ sample makes not selectable the first visible item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetSelectableItem( items.GetFirstVisibleItem(), FALSE );
```

The following VB.NET sample makes not selectable the first visible item:

```
With AxTree1.Items
    .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C# sample makes not selectable the first visible item:

```
axTree1.Items.set_SelectableItem(axTree1.Items.FirstVisibleItem, false);
```

The following VFP sample makes not selectable the first visible item:

```
with thisform.Tree1.Items
    .DefaultItem = .FirstVisibleItem
    .SelectableItem(0) = .f.
endwith
```

method Items.SelectAll ()

Selects all items.

Type	Description

Use the SelectAll method to select all visible items in the tree. The SelectAll method has effect only if the SingleSel property is False, if the control supports multiple items selection. Use the UnselectAll method to unselect all items in the list. Use the SelectItem property to select or unselect a specified item. Use the SelectedItem property to retrieve a value that indicates whether the item is selected or unselected. Use the SelectCount property to retrieve the number of selected items

property Items.SelectCount as Long

Counts the number of items that are selected into 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 SelectCount property gets 0, if no items are selected in the control. The ExTree control supports multiple selection. Use the SingleSel property of the control to allow multiple selection. Use the SelectedItem property to retrieve the handle of the selected item(s). The control fires the SelectionChanged event when user changes the selection in the control. Use the SelectItem property to select programmatically an item. Use the SelForeColor and SelBackColor properties to specify colors for selected items. If the control supports only single selection (SingleSel property is True), the FocusItem retrieves the selected item too. Use the SelectAll method to select all visible items in the tree. Use the UnselectAll method to unselect all items in the control.

If the control's SingleSel is false, then the following statement retrieves the handle for the selected item: Tree1.Items.SelectedItem().

If the control supports multiple selection then the following VB sample shows how to enumerate all selected items:

```
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long, nSels As Long
nCols = Tree1.Columns.Count
With Tree1.Items
   nSels = .SelectCount
   For i = 0 To nSels - 1
      Dim s As String
      For j = 0 To nCols - 1
         s = s + .CellCaption(.SelectedItem(i), j) + Chr(9)
      Next
      Debug.Print s
   Next
End With
```

The following VB sample unselects all items in the control:

```
With Tree1
   .BeginUpdate
   With .Items
      While Not .SelectCount = 0
         .SelectItem(.SelectedItem(0)) = False
      Wend
   End With
   .EndUpdate
End With
```

The following C++ sample enumerates the selected items:

```
CItems items = m_tree.GetItems();
long n = items.GetSelectCount();
if ( n != 0 )
{
   for ( long i = 0; i < n; i++ )
   {
      long h = items.GetSelectedItem( i );
      COleVariant vtString;
      vtString.ChangeType( VT_BSTR, &items.GetCellCaption( COleVariant( h ), COleVariant(
(long)0 ) ) );
      CString str = V_BSTR( &vtString );
      MessageBox( str );
   }
}
```

The following C++ sample unselects all items in the control:

```
m_tree.BeginUpdate();
CItems items = m_tree.GetItems();
while ( items.GetSelectCount() )
   items.SetSelectItem( items.GetSelectedItem( 0 ), FALSE );
m_tree.EndUpdate();
```

The following VB.NET sample enumerates the selected items:

```
With AxTree1.Items
```

```
    Dim nCols As Integer = AxTree1.Columns.Count, i As Integer
    For i = 0 To .SelectCount - 1
        Debug.Print(.CellCaption(.SelectedItem(i), 0))
    Next
End With
```

The following VB.NET sample unselects all items in the control:

```
With AxTree1
    .BeginUpdate()
    With .Items
        While Not .SelectCount = 0
            .SelectItem(.SelectedItem(0)) = False
        End While
    End With
    .EndUpdate()
End With
```

The following C# sample enumerates the selected items:

```
for (int i = 0; i < axTree1.Items.SelectCount; i++)
{
    object strCaption = axTree1.Items.get_CellCaption(axTree1.Items.get_SelectedItem(i), 0);
    System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following C# sample unselects all items in the control:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
while (items.SelectCount != 0)
    items.set_SelectItem(items.get_SelectedItem(0), false);
axTree1.EndUpdate();
```

The following VFP sample enumerates the selected items:

```
with thisform.Tree1.Items
    local i
    for i = 0 to .SelectCount - 1
```

```
      .DefaultItem = .SelectedItem(i)
      wait window nowait .CellCaption(0,0)
   next
endwith
```

The following VFP sample unselects all items in the control:

```
With thisform.Tree1
   .BeginUpdate()
   with .Items
      do while ( .SelectCount() # 0 )
         .DefaultItem = .SelectedItem(0)
         .SelectItem(0) = .f.
      enddo
   endwith
   .EndUpdate()
EndWith
```

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 the selected items collection.
HITEM	A long expression that indicates the handle of the selected item.

Use the SelectedItem property to get the handle of the selected item(s) in the control. Use the SelectCount property to find out how many items are selected in the control. The control fires the SelectionChanged event when user changes the selection in the control. Use the SelectItem property to select programmatically an item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. Use the SingleSel property to enable single or multiple selection. Use the SelForeColor and SelBackColor properties to specify colors for selected items.

The following sample shows hot to print the caption for the selected item: Debug.Print Tree1.Items.CellCaption(Tree1.Items.SelectedItem(0), 0).

The following sample applies an italic font attribute to the selected item:

```
Private Sub Tree1_SelectionChanged()
  If Not (h = 0) Then Tree1.Items.ItemItalic(h) = False
  h = Tree1.Items.SelectedItem()
  Tree1.Items.ItemItalic(h) = True
End Sub
```

The following VB sample enumerates the selected items:

```
Dim i As Long
With Tree1.Items
  For i = 0 To .SelectCount - 1
    Debug.Print .CellCaption(.SelectedItem(i), 0)
  Next
End With
```

The following VB sample unselects all items in the control:

```
With Tree1
   .BeginUpdate
   With .Items
      While Not .SelectCount = 0
         .SelectItem(.SelectedItem(0)) = False
      Wend
   End With
   .EndUpdate
End With
```

The following VC sample displays the selected items:

```
#include "Items.h"

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}

CItems items = m_tree.GetItems();
for ( long i = 0; i < items.GetSelectCount(); i++ )
{
   COleVariant vtItem( items.GetSelectedItem( i ) );
   CString strOutput;
   strOutput.Format( "%s\n", V2S( &items.GetCellCaption( vtItem, COleVariant( (long)0 ) ) )
);
   OutputDebugString( strOutput );
}
```

The following C++ sample unselects all items in the control:

```
m_tree.BeginUpdate();
CItems items = m_tree.GetItems();
while ( items.GetSelectCount() )
   items.SetSelectItem( items.GetSelectedItem( 0 ), FALSE );
m_tree.EndUpdate();
```

The following VB.NET sample displays the selected items:

```
With AxTree1.Items
   Dim i As Integer
   For i = 0 To .SelectCount - 1
      Debug.WriteLine(.CellCaption(.SelectedItem(i), 0))
   Next
End With
```

The following VB.NET sample unselects all items in the control:

```
With AxTree1
   .BeginUpdate()
   With .Items
      While Not .SelectCount = 0
         .SelectItem(.SelectedItem(0)) = False
      End While
   End With
   .EndUpdate()
End With
```

The following C# sample displays the selected items:

```
for ( int i = 0; i < axTree1.Items.SelectCount - 1; i++ )
{
   object cell = axTree1.Items.get_CellCaption( axTree1.Items.get_SelectedItem( i), 0 );
   System.Diagnostics.Debug.WriteLine( cell != null ? cell.ToString() : "" );
}
```

The following C# sample unselects all items in the control:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
while (items.SelectCount != 0)
    items.set_SelectItem(items.get_SelectedItem(0), false);
axTree1.EndUpdate();
```

The following VFP sample displays the selected items:

```
with thisform.Tree1.Items
    for i = 0 to .SelectCount - 1
        .DefaultItem = .SelectedItem( i )
        wait window nowait .CellCaption( 0, 0 )
    next
endwith
```

The following VFP sample unselects all items in the control:

```
With thisform.Tree1
    .BeginUpdate()
    with .Items
        do while ( .SelectCount() # 0 )
            .DefaultItem = .SelectedItem(0)
            .SelectItem(0) = .f.
        enddo
    endwith
    .EndUpdate()
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 item's handle that is selected or unselected.
Boolean	A boolean expression that indicates the item's state. True if the item is selected, and False if the item is not selected.

Use the SelectItem to select or unselect a specified item (that's selectable). Use the SelectableItem property to specify the user can select an item. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the FocusItem property to get the focused item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. The control fires the SelectionChanged event when user selects an item. Use the SelForeColor and SelBackColor properties to specify colors for selected items. Use the SingleSel property to allow multiple selection. Use the SelectPos property to select an item giving its position. Use the EnsureVisibleItem property to ensure that an item is visible. Use the SelectAll method to select all visible items in the tree. Use the UnselectAll method to unselect all items in the control.

The following VB sample shows how to select the first created item:
Tree1.Items.SelectItem(Tree1.Items(0)) = True

The following VB sample selects the first visible item:

```
With Tree1.Items
   .SelectItem(.FirstVisibleItem) = True
End With
```

The following VB sample enumerates the selected items:

```
Dim i As Long
With Tree1.Items
   For i = 0 To .SelectCount - 1
      Debug.Print .CellCaption(.SelectedItem(i), 0)
   Next
End With
```

The following C++ sample selects the first visible item:

```
#include "Items.h"
CItems items = m_tree.GetItems();
items.SetSelectItem( items.GetFirstVisibleItem(), TRUE );
```

The following C++ sample unselects all items in the control:

```
m_tree.BeginUpdate();
CItems items = m_tree.GetItems();
while ( items.GetSelectCount() )
   items.SetSelectItem( items.GetSelectedItem( 0 ), FALSE );
m_tree.EndUpdate();
```

The following VB.NET sample selects the first visible item:

```
With AxTree1.Items
   .SelectItem(.FirstVisibleItem) = True
End With
```

The following VB.NET sample unselects all items in the control:

```
With AxTree1
   .BeginUpdate()
   With .Items
      While Not .SelectCount = 0
         .SelectItem(.SelectedItem(0)) = False
      End While
   End With
   .EndUpdate()
End With
```

The following C# sample selects the first visible item:

```
axTree1.Items.set_SelectItem(axTree1.Items.FirstVisibleItem, true);
```

The following C# sample unselects all items in the control:

```
axTree1.BeginUpdate();
EXTREELib.Items items = axTree1.Items;
```

```
while (items.SelectCount != 0)
   items.set_SelectItem(items.get_SelectedItem(0), false);
axTree1.EndUpdate();
```

The following VFP sample selects the first visible item:

```
with thisform.Tree1.Items
   .DefaultItem = .FirstVisibleItem
   .SelectItem(0) = .t.
endwith
```

The following VFP sample unselects all items in the control:

```
With thisform.Tree1
   .BeginUpdate()
   with .Items
      do while ( .SelectCount() # 0 )
         .DefaultItem = .SelectedItem(0)
         .SelectItem(0) = .f.
      enddo
   endwith
   .EndUpdate()
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 handle of the item being moved.
NewParent as HITEM	A long expression that indicates the handle of the new parent item.

Use the SetParent property to change the parent item at runtime. Use the InsertItem property to insert child items. Use the InsertControlItem property to insert ActiveX controls. 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: Tree1.Items.SetParent Tree1.Items(0), Tree1.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. Thought, the position of an unsortable item can be changed using the ItemPosition property. Use the SortableItem to specify a group item, a total item or a separator item. An unsortable item is not counted by a total field. The SortType property specifies the type of repositioning is being applied on the column when a sort operation is performed. The SortOrder property specifies whether the column is sorted ascendant or descendent. Use the 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 HITEM	A long expression that indicates the item's handle that is going to be sorted.
ColIndex as Variant	A long expression that indicates the column's index or the cell's handle, a string expression that indicates the column's caption.
Ascending as Boolean	A boolean expression that defines the sort order.

The SortChildren will not recurse through the tree, 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: Tree1.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, depending on the SortOnClick property. The SortOrder property sorts the items and displays the sorting icon in the column's header. Use the AllowSort property to avoid sorting a column when the user clicks the column.

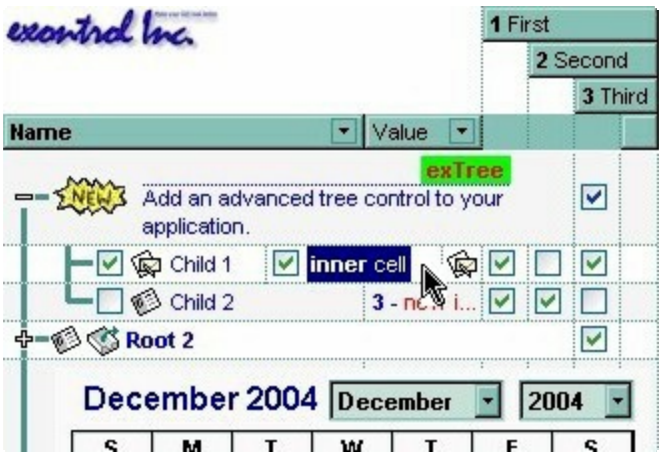
property Items.SplitCell ([Item as Variant], [ColIndex as Variant]) as Variant

Splits a cell, and returns the inner created cell.

Type	Description
Item as Variant	A long expression that indicates the handle of the item where a cell is being divided, or 0. If the Item parameter is 0, the ColIndex parameter must indicate the handle of the cell.
ColIndex as Variant	A long expression that indicates the index of the column where a cell is divided, or a long expression that indicates the handle of the cell being divided, if the Item parameter is missing or it is zero.
Variant	A long expression that indicates the handle of the cell being created.

The SplitCell method splits a cell in two cells. The newly created cell is called inner cell. The SplitCell method always returns the handle of the inner cell. If the cell is already divided using the SplitCell method, it returns the handle of the inner cell without creating a new inner cell. You can split an inner cell too, and so you can have a master cell divided in multiple cells. Use the CellWidth property to specify the width of the inner cell. Use the CellCaption property to assign a caption to a cell. Use the InnerCell property to access an inner cell giving its index. Use the CellParent property to get the parent of the inner cell. Use the CellItem property to get the owner of the cell. Use the UnsplitCell method to remove the inner cell if it exists. Use the MergeCells property to combine two or more cells in a single cell. Use the SelectableItem property to specify the user can select an item.

("Merge" means multiple cells in a single cell, "Split" means multiple cells **inside** a single cell)



The following VB sample splits a single cell in two cells (Before running the following

sample, please make sure that your control contains columns, and at least an item):

```
With Tree1.Items
   Dim h As HITEM, f As HCELL
   h = .FirstVisibleItem
   f = .SplitCell(h, 0)
   .CellCaption(, f) = "inner cell"
End With
```

The following C++ sample splits the first visible cell in two cells:

```
#include "Items.h"
CItems items = m_tree.GetItems();
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
COleVariant vtSplit = items.GetSplitCell( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
items.SetCellCaption( vtMissing, vtSplit, COleVariant( "inner cell" ) );
```

The following VB.NET sample splits the first visible cell in two cells:

```
With AxTree1.Items
   Dim i As Object
   i = .SplitCell(.FirstVisibleItem, 0)
   .CellCaption(Nothing, i) = "inner cell"
End With
```

The following C# sample splits the first visible cell in two cells:

```
EXTREELib.Items items = axTree1.Items;
object i = items.get_SplitCell(items.FirstVisibleItem, 0);
items.set_CellCaption(null, i, "inner cell");
```

The following VFP sample splits the first visible cell in two cells:

```
with thisform.Tree1.Items
   local i
   i = .SplitCell(.FirstVisibleItem,0)
   local s, crlf
   crlf = chr(13) + chr(10)
```

```
    s = "Items" + crlf
    s = s + "{" + crlf
    s = s + "CellCaption(," + str(i) + ") = " + chr(34) + "inner cell" + chr(34) + crlf
    s = s + "}"
    thisform.Tree1.Template = s
endwith
```

method Items.UnmergeCells ([Cell as Variant])

Unmerges a list of cells.

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 samples show few methods to unmerge cells:

```
With Tree1
   With .Items
      .UnmergeCells .ItemCell(.RootItem(0), 0)
   End With
End With
```

```
With Tree1
   With .Items
      Dim r As Long
      r = .RootItem(0)
      .UnmergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1))
   End With
End With
```

```
With Tree1
   .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 UnselectAll method to unselect all items in the list. The UnselectAll method has effect only if the SingleSel property is False, if the control supports multiple items selection. Use the SelectAll method to select all items in the list. Use the SelectItem property to select or unselect a specified item. Use the SelectedItem property to retrieve a value that indicates whether the item is selected or unselected. Use the SelectCount property to retrieve the number of selected items. The RemoveSelection method removes the selected items (including the descendents).

method Items.UnsplitCell ([Item as Variant], [ColIndex as Variant])

Unsplits a cell.

Type	Description
Item as Variant	A long expression that indicates the handle of the item, or 0. If the Item parameter is 0, the ColIndex parameter must indicate the handle of the cell.
ColIndex as Variant	A long expression that indicates the index of the column where a cell is divided, or a long expression that indicates the handle of the cell being divided, if the Item parameter is missing or it is zero.

Use the UnsplitCells method to remove the inner cells. The SplitCell method splits a cell in two cells, and retrieves the newly created cell. The UnsplitCell method has no effect if the cell contains no inner cells. The UnplitCells method remove recursively all inner cells. For instance, if a cell contains an inner cell, and this inner cell contains another inner cell, when calling the UnplitCells method for the master cell, all inner cells inside of the cell will be deleted. Use the CellParent property to get the parent of the inner cell. Use the CellItem property to get the owner of the cell. Use the InnerCell property to access an inner cell giving its index. Use the UnmergeCells method to unmerge merged cells. ("Merge" means multiple cells in a single cell, "Split" means multiple cells **inside** a single).

property Items.VisibleCount as Long

Retrieves the number of visible items.

Type	Description
Long	Counts the visible items.

Use FirstVisibleItem and NextVisibleItem properties to determine the items that fit the client area. Use the IsItemVisible property to check whether an item fits the control's client area. Use the 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 munus 1, indicates the number of visible items, and there is a filter applied (match found)

The VisibleCount property retrieves the number of items being displayed in the control's client area. Use FirstVisibleItem and NextVisibleItem properties to determine the items being displayed in the control's client area. Use the IsItemVisible property to check whether an item fits the control's client area. Use the ItemCount property to count the items in the control. Use the ChildCount property to count the child items

OleEvent object

The OleEvent object holds information about an event fired by an ActiveX control hosted by in item that was created using the InsertControlItem method.

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

property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

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

The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent is fired.

```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
  Debug.Print "Event name:" & Ev.Name
  If (Ev.CountParam = 0) Then
    Debug.Print "The event has no arguments."
  Else
    Debug.Print "The event has the following arguments:"
    Dim i As Long
    For i = 0 To Ev.CountParam - 1
      Debug.Print Ev(i).Name; " = " & Ev(i).Value
    Next
  End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

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 OnItemOleEventTree1(long Item, LPDISPATCH Ev)
{
    EXTREELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXTREELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}
```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXTREELib namespace that include all objects and types of the control's TypeLibrary. In case your extree.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxTree1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles AxTree1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
```

```
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXTREELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
private void axTree1_ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXTREELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an item:

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

property OleEvent.ID as Long

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

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

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

property OleEvent.Name as String

Retrieves the original name of the fired event.

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

Use the ID property to specify a specified even by its identifier. Use the ToString property to display information about fired event such us name, parameters, types and values. Use the CountParam property to count the parameters of an OLE event. Use the Param property to get the event's parameter. Use the Value property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent is fired.

```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
  Debug.Print "Event name:" & Ev.Name
  If (Ev.CountParam = 0) Then
    Debug.Print "The event has no arguments."
  Else
    Debug.Print "The event has the following arguments:"
    Dim i As Long
    For i = 0 To Ev.CountParam - 1
      Debug.Print Ev(i).Name; " = " & Ev(i).Value
    Next
  End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

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 OnItemOleEventTree1(long Item, LPDISPATCH Ev)
{
    EXTREELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXTREELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}
```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXTREELib namespace that include all objects and types of the control's TypeLibrary. In case your extree.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxTree1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles AxTree1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXTREELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
private void axTree1_ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXTREELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an item:

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

property OleEvent.Param (Item as Variant) as OleEventParam

Retrieves an OleEventParam object given either the index of the parameter, or its name.

Type	Description
Item as Variant	A long expression that indicates the argument's index or a string expression that indicates the argument's name.
OleEventParam	An OleEventParam object that contains the name and the value for the argument.

The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent is fired.

```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
  Debug.Print "Event name:" & Ev.Name
  If (Ev.CountParam = 0) Then
    Debug.Print "The event has no arguments."
  Else
    Debug.Print "The event has the following arguments:"
    Dim i As Long
    For i = 0 To Ev.CountParam - 1
      Debug.Print Ev(i).Name; " = " & Ev(i).Value
    Next
  End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

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 OnItemOleEventTree1(long Item, LPDISPATCH Ev)
{
    EXTREELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXTREELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}
```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXTREELib namespace that include all objects and types of the control's TypeLibrary. In case your extree.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxTree1_ItemOleEvent(ByVal sender As Object, ByVal e As
```

```
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles AxTree1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXTREELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
private void axTree1_ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXTREELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an item:

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

property OleEvent.ToString as String

Retrieves information about the event.

Type	Description
String	A String expression that shows information about an OLE event. The ToString property gets the information as follows: Name[ID] (Param/Type = Value, Param/Type = Value, ...). For instance, "KeyDown[-602] (KeyCode/Short* = 9,Shift/Short = 0)" indicates that the KeyDown event is fired, with the identifier -602 with two parameters KeyCode as a reference to a short type with the value 8, and Shift parameter as Short type with the value 0.

Use the ToString property to display information about fired event such us name, parameters, types and values. Using the ToString property you can quickly identifies the event that you should handle in your application. Use the ID property to specify a specified even by its identifier. Use the Name property to get the name of the event. Use the Param property to access a specified parameter using its index or its name.

Displaying ToString property during the OLE Event event may show data like follows:

```
MouseMove[-606](Button/Short = 0,Shift/Short = 0,X/Long = 46,Y/Long = 15)
MouseDown[-605](Button/Short = 1,Shift/Short = 0,X/Long = 46,Y/Long = 15)
KeyDown[-602](KeyCode/Short* = 83,Shift/Short = 0)
KeyPress[-603](KeyAscii/Short* = 115)
Change[2]()
KeyUp[-604](KeyCode/Short* = 83,Shift/Short = 0)
MouseUp[-607](Button/Short = 1,Shift/Short = 0,X/Long = 46,Y/Long = 15)
MouseMove[-606](Button/Short = 0,Shift/Short = 0,X/Long = 46,Y/Long = 15)
```

OleEventParam object

The OleEventParam holds the name and the value for an event's argument.

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

property OleEventParam.Name as String

Retrieves the name of the event's parameter.

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

The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent is fired.

```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
  Debug.Print "Event name:" & Ev.Name
  If (Ev.CountParam = 0) Then
    Debug.Print "The event has no arguments."
  Else
    Debug.Print "The event has the following arguments:"
    Dim i As Long
    For i = 0 To Ev.CountParam - 1
      Debug.Print Ev(i).Name; " = " & Ev(i).Value
    Next
  End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

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 OnItemOleEventTree1(long Item, LPDISPATCH Ev)
{
    EXTREELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXTREELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}
```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXTREELib namespace that include all objects and types of the control's TypeLibrary. In case your extree.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxTree1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles AxTree1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
```

```
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXTREELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
private void axTree1_ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXTREELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an item:

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

property OleEventParam.Value as Variant

Retrieves or sets the value of the event's parameter.

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

The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent is fired.

```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
  Debug.Print "Event name:" & Ev.Name
  If (Ev.CountParam = 0) Then
    Debug.Print "The event has no arguments."
  Else
    Debug.Print "The event has the following arguments:"
    Dim i As Long
    For i = 0 To Ev.CountParam - 1
      Debug.Print Ev(i).Name; " = " & Ev(i).Value
    Next
  End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

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 OnItemOleEventTree1(long Item, LPDISPATCH Ev)
{
    EXTREELib::IOleEventPtr spEvent( Ev );
    CString strOutput;
    strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
    OutputDebugString( strOutput );
    if ( spEvent->CountParam == 0 )
        OutputDebugString( "The event has no parameters." );
    else
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXTREELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}
```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXTREELib namespace that include all objects and types of the control's TypeLibrary. In case your extree.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxTree1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles AxTree1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
```

```
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXTREELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
private void axTree1_ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXTREELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an item:

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

Tree 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: {3C5FC763-72BA-4B97-9985-81862E9251F2}. The object's program identifier is: "Exontrol.Tree". The /COM object module is: "ExTree.dll"

The ExTree component provides the entire range of capabilities you would expect to see in a state-of-the-art Tree component. The exTree control simulates a simple tree, a multi-column tree, a list or a listview control. Features include: skinnable interface, **ActiveX hosting** (you can place any ActiveX component in any item of the tree), events from contained components are fired through to your program using the exact same model used in VB6 for components added at run time, ADO and DAO support, multiple columns, sorting, multiple selection, user resizable columns, columns dragable, locked or unlocked columns, cells can be formatted individually, or via columns or rows, radio buttons, images, check boxes, multi-line list items, incremental search feature, mouse wheel support, column alignment right, left, or center, format columns, and more. The ExTree component supports the following methods and properties:

Name	Description
AllowCopyTemplate	Specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.
AllowEdit	Retrieves or sets a value that indicates whether the editing tree is allowed or disabled.
AllowSelectNothing	Specifies whether the current selection is erased, once the user clicks outside of the items section.
AnchorFromPoint	Retrieves the identifier of the anchor from point.
Appearance	Retrieves or sets the control's appearance.
ApplyFilter	Applies the filter.
ASCIILower	Specifies the set of lower characters.
ASCIIUpper	Specifies the set of upper characters.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoDrag	Gets or sets a value that indicates the way the component supports the AutoDrag feature.
AutoSearch	Enables or disables the auto search feature.
BackColor	Retrieves or sets a value that indicates the control's background color.
	Specifies the background color used to display alternate

BackColorAlternate	items in the control.
BackColorHeader	Specifies the header's background color.
BackColorLevelHeader	Specifies the multiple levels header's background color.
BackColorLock	Retrieves or sets a value that indicates the control's background color for the locked area.
BackColorSortBar	Retrieves or sets a value that indicates the sort bar's background color.
BackColorSortBarCaption	Returns or sets a value that indicates the caption's background color in the control's sort bar.
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.
BorderStyle	Retrieves or sets the border style of the control.
CheckImage	Retrieves or sets a value that indicates the image used by cells of checkbox type.
ClearFilter	Clears the filter.
ColumnAutoResize	Returns or sets a value indicating whether the control will automatically size its visible columns to fit on the control's client width.
ColumnFromPoint	Retrieves the column from point.
Columns	Retrieves the control's column collection.
ColumnsAllowSizing	Retrieves or sets a value that indicates whether a user can resize columns at run-time.
ColumnsFloatBarSortOrder	Specifies the sorting order for the columns being shown in the control's columns floating panel.
ColumnsFloatBarVisible	Retrieves or sets a value that indicates whether the the columns float bar is visible or hidden.
ConditionalFormats	Retrieves the conditional formatting collection.
ContinueColumnScroll	Retrieves or sets a value indicating whether the control will automatically scroll the visible columns by pixel or by column width.
Copy	Copies the control's content to the clipboard, in the EMF format.

CopyTo	Exports the control's view to an EMF file.
CountLockedColumns	Retrieves or sets a value indicating the number of locked columns. A locked column is not scrollable.
DataSource	Retrieves or sets a value that indicates the data source for object.
DefaultItemHeight	Retrieves or sets a value that indicates the default item height.
Description	Changes descriptions for control objects.
DetectAddNew	Specifies whether the control detects when a new record is added to the bounded recordset.
DrawGridLines	Retrieves or sets a value that indicates whether the grid lines are visible or hidden.
Enabled	Enables or disables the control.
EndUpdate	Resumes painting the control after painting is suspended by the BeginUpdate method.
EnsureOnSort	Specifies whether the control ensures that the focused item fits the control's client area, when the user sorts the items.
EnsureVisibleColumn	Scrolls the control's content to ensure that the column fits the client area.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
ExpandOnDblClick	Specifies whether the item is expanded or collapsed if the user dbl clicks the item.
ExpandOnKeys	Specifies a value that indicates whether the control expands or collapses a node when user presses arrow keys.
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 filter bar is automatically resized to fit its description.
FilterBarPrompt	Specifies the caption to be displayed when the filter pattern is missing.
FilterBarPromptColumns	Specifies the list of columns to be used when filtering using the prompt.
FilterBarPromptPattern	Specifies the pattern for the filter prompt.
FilterBarPromptType	Specifies the type of the filter prompt.
FilterBarPromptVisible	Shows or hides the filter prompt.
FilterCriteria	Retrieves or sets the filter criteria.
FilterInclude	Specifies the items being included after the user applies the filter.
Font	Retrieves or sets the control's font.
ForeColor	Retrieves or sets a value that indicates the control's foreground color.
ForeColorHeader	Specifies the header's foreground color.
ForeColorLock	Retrieves or sets a value that indicates the control's foreground color for the locked area.
ForeColorSortBar	Retrieves or sets a value that indicates the sort bar's foreground color.
FormatABC	Formats the A,B,C values based on the giving expression and returns the result.
FormatAnchor	Specifies the visual effect for anchor elements in HTML captions.
FreezeEvents	Prevents the control to fire any event.
FullRowSelect	Enables full-row selection in the control.
GetItems	Gets the collection of items into a safe array,
GridLineColor	Specifies the grid line color.
GridLineStyle	Specifies the style for gridlines in the list part of the control.
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.
HeaderEnabled	Enables or disables the control's header.
HeaderHeight	Retrieves or sets a value indicating the 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 tree's header is visible or hidden.
HideSelection	Returns a value that determines whether selected item appears highlighted when a control loses the focus.
HotBackColor	Retrieves or sets a value that indicates the hot-tracking background color.
HotForeColor	Retrieves or sets a value that indicates the hot-tracking foreground color.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
HyperLinkColor	Specifies the hyperlink color.
Images	Sets at runtime the control's image list. The Handle should be a handle to an Image List Control.
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.
ItemFromPoint	Retrieves the item from point.
Items	Retrieves the control's item collection.
ItemsAllowSizing	Retrieves or sets a value that indicates whether a user can resize items at run-time.
Layout	Saves or loads the control's layout, such as positions of the columns, scroll position, filtering values.

LinesAtRoot	Link items at the root of the hierarchy.
MarkSearchColumn	Retrieves or sets a value that indicates whether the searching column is marked or unmarked
OLEDrag	Causes a component to initiate an OLE drag/drop operation.
OLEDropMode	Returns or sets how a target component handles drop operations
Picture	Retrieves or sets a graphic to be displayed in the control.
PictureDisplay	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background
PictureDisplayLevelHeader	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's header background.
PictureLevelHeader	Retrieves or sets a graphic to be displayed in the control's header when multiple levels is on.
PutItems	Adds an array of integer, long, date, string, double, float, or variant arrays to the control.
RadioImage	Retrieves or sets a value that indicates the image used by cells of radio type.
RClickSelect	Retrieves or sets a value that indicates whether an item is selected using right mouse button.
Refresh	Refreshes the control's content.
RemoveSelection	Removes the selected items (including the descendents)
ReplaceIcon	Adds a new icon, replaces an icon or clears the control's image list.
RightToLeft	Indicates whether the component should draw right-to-left for RTL languages.
Scroll	Scrolls the control's content.
ScrollBars	Returns or sets a value that determines whether the control has horizontal and/or vertical scroll bars.
ScrollButtonHeight	Specifies the height of the button in the vertical scrollbar.
ScrollButtonWidth	Specifies the width of the button in the horizontal scrollbar.
ScrollBySingleLine	Retrieves or sets a value that indicates whether the control scrolls the lines to the end. If you have at least a cell that has SingleLine false, you have to check the ScrollBySingleLine property..
ScrollFont	Retrieves or sets the scrollbar's font.

ScrollHeight	Specifies the height of the horizontal scrollbar.
ScrollOrderParts	Specifies the order of the buttons in the scroll bar.
ScrollPartCaption	Specifies the caption being displayed on the specified scroll part.
ScrollPartCaptionAlignment	Specifies the alignment of the caption in the part of the scroll bar.
ScrollPartEnable	Indicates whether the specified scroll part is enabled or disabled.
ScrollPartVisible	Indicates whether the specified scroll part is visible or hidden.
ScrollPos	Specifies the vertical/horizontal scroll position.
ScrollThumbSize	Specifies the size of the thumb in the scrollbar.
ScrollToolTip	Specifies the tooltip being shown when the user moves the scroll box.
ScrollWidth	Specifies the width of the vertical scrollbar.
SearchColumnIndex	Retrieves or sets a value indicating the column's index that is used for auto search feature.
SelBackColor	Retrieves or sets a value that indicates the selection background color.
SelBackMode	Retrieves or sets a value that indicates whether the selection is transparent or opaque.
SelectColumn	Specifies whether the user selects cells only in SelectColumnIndex column, while FullRowSelect property is False.
SelectColumnIndex	Retrieves or sets a value that indicates control column's index where the user is able to select an item. It has effect only for FullRowSelect = false.
SelectColumnInner	Retrieves or sets a value that indicates the index of the inner cell that's selected.
SelectOnRelease	Indicates whether the selection occurs when the user releases the mouse button.
SelForeColor	Retrieves or sets a value that indicates the selection foreground color.
SelLength	Returns or sets the number of characters selected.
SelStart	Returns or sets the starting point of text selected; indicates the position of the insertion point if no text is

	selected.
ShowFocusRect	Retrieves or sets a value indicating whether the control draws a thin rectangle arround 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 the control displays the locked items.
ShowToolTip	Shows the specified tooltip at given position.
SingleSel	Retrieves or sets a value that indicates whether the control supports single or multiple selection.
SingleSort	Returns or sets a value that indicates whether the control supports sorting by single or multiple columns.
SortBarCaption	Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns.
SortBarColumnWidth	Specifies the maximum width a column can be in the control's sort bar.
SortBarHeight	Retrieves or sets a value that indicates the height of the control's sort bar.
SortBarVisible	Retrieves or sets a value that indicates whether control's sort bar is visible or hidden.
SortOnClick	Retrieves or sets a value that indicates whether the control sorts automatically the data when the user click on column's caption.
Statistics	Gives statistics data of objects being hold by the control.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
TemplatePut	Defines inside variables for the next Template/ExecuteTemplate call.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipMargin	Defines the size of the control's tooltip margins.
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.
ToTemplate	Generates the control's template.
TreeColumnIndex	Retrieves or sets a value indicating the column's index where the hierarchy will be displayed.
UseTabKey	Specifies whether the TAB key is used to change the searching column.
UseVisualTheme	Specifies whether the control uses the current visual theme to display certain UI parts.
Version	Retrieves the control's version.
VisualAppearance	Retrieves the control's appearance.
VisualDesign	Invokes the control's VisualAppearance designer.

property Tree.AllowCopyTemplate as Boolean

Specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.

Type	Description
Boolean	A Boolean expression that indicates whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.

By default, the AllowCopyTemplate property is True, only for trial-demo version, and False, for the registered version. So, by default, the Shift + Ctrl + Alt + Insert sequence is working in the trial version, and it doesn't work on the registered version. Use the Version property to find out what version of the control you are running. Use the AllowCopyTemplate property for debugging purpose. Use the AllowCopyTemplate property to easily copy the control's content to the clipboard, as template form, and so you can send us a sample without being necessary to send the entire sample to us. The AllowCopyTemplate property is not serialized in the form's persistence, so you need to set it in the code for a particular value. If the AllowCopyTemplate property is True, the user may use the Shift + Ctrl + Alt + Insert sequence to copy the control's content to the clipboard, in template form. If the control manages to copy the control's content to the clipboard, you should hear a beep. The property uses the ToTemplate property to generate the control's template, at runtime. The format of the clipboard being copied is plain text. Use the Template property to apply the generated template to an empty control.

property Tree.AllowEdit as Boolean

Retrieves or sets a value that indicates whether the editing tree is allowed or disabled.

Type	Description
Boolean	A boolean expression that indicates whether the editing tree is allowed or disabled.

By default, the AllowEdit property is false. If the AllowEdit property is True, the control fires the BeforeCellEdit event just before editing a cell, and fires the AfterCellEdit after that edit operation ends. Use the Edit method to pragmatically edit an item. Use the SelStart and SelLenght properties to specify the selected text when edit operation starts.

property Tree.AllowSelectNothing as Boolean

Specifies whether the current selection is erased, once the user clicks outside of the items section.

Type	Description
Boolean	A Boolean expression that specifies whether the current selection is erased, once the user clicks outside of the items section.

By default, the AllowSelectNothing property is False. The AllowSelectNothing property specifies whether the current selection is erased, once the user clicks outside of the items section. For instance, if the control's SingleSel property is True, and AllowSelectNothing property is True, you can un-select the single-selected item if pressing the CTRL + Space, or by CTRL + click.

property Tree.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 Tree1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With Tree1
    .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 AxTree1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseMoveEvent) Handles AxTree1.MouseMoveEvent
  With AxTree1
    .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 axTree1_MouseMoveEvent(object sender,
AxEXTREELib._ITreeEvents_MouseMoveEvent e)
{
    axTree1.ShowToolTip(axTree1.get_AnchorFromPoint(-1, -1));
}
```

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

```
void OnMouseMoveTree1(short Button, short Shift, long X, long Y)
{
    COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
    m_tree.ShowToolTip( m_tree.GetAnchorFromPoint( -1, -1 ), vtEmpty, vtEmpty, vtEmpty );
}
```

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

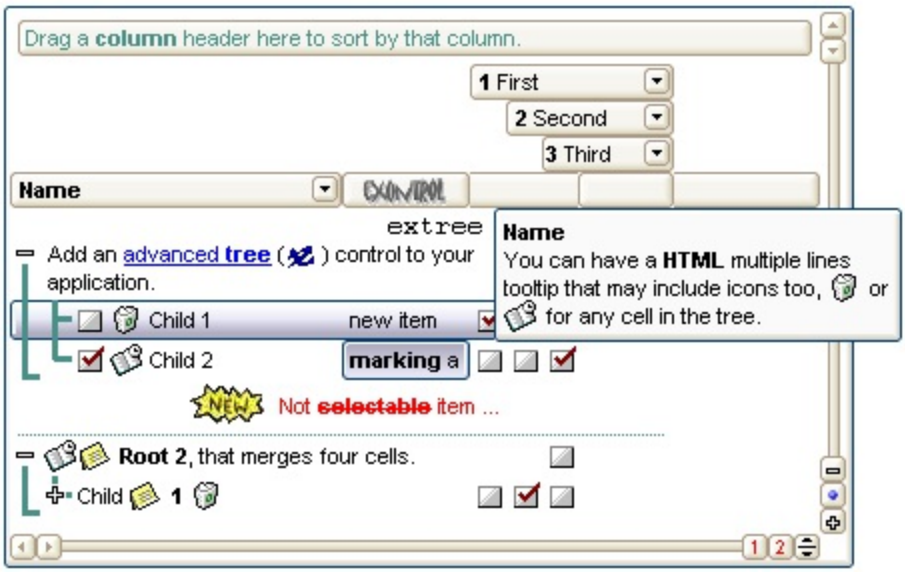
```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform
    With .Tree1
        .ShowToolTip(.AnchorFromPoint(-1, -1))
    EndWith
endwith
```

property Tree.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the Appearance collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. ***The Client object in the skin, defines the client area of the control. The list/hierarchy, scrollbars are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The frame.ebn file contains such of objects. Use the eXButton's Skin builder to view or change this file***

Use the Appearance property to specify the control's border. Use the HeaderAppearance property to change the control's header bar appearance. Use the Add method to add new skins to the control. Use the BackColor property to specify the control's background color. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips.



The following VB sample changes the visual aspect of the borders of the control (please check the above picture for round corners):

```
With Tree1
```

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

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

```
With AxTree1
  .BeginUpdate()
  .VisualAppearance.Add(&H16, "c:\temp\frame.ebn")
  .Appearance = &H16000000
  .BackColor = Color.FromArgb(250, 250, 250)
  .EndUpdate()
End With
```

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

```
axTree1.BeginUpdate();
axTree1.VisualAppearance.Add(0x16, "c:\\temp\\frame.ebn");
axTree1.Appearance = (EXTREELib.AppearanceEnum)0x16000000;
axTree1.BackColor = Color.FromArgb(250, 250, 250);
axTree1.EndUpdate();
```

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

```
m_tree.BeginUpdate();
m_tree.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\frame.ebn" ) );
m_tree.SetAppearance( 0x16000000 );
m_tree.SetBackColor( RGB(250,250,250) );
m_tree.EndUpdate();
```

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

```
with thisform.Tree1
  .BeginUpdate
    .VisualAppearance.Add(0x16, "c:\temp\frame.ebn")
    .Appearance = 0x16000000
```

```
    .BackColor = RGB(250, 250, 250)
  .EndUpdate
endwith
```

method Tree.ApplyFilter ()

Applies the filter.

Type	Description

The ApplyFilter method updates the control's content once that user sets the filter using the Filter and FilterType properties. Use the ClearFilter method to clear the control's filter. Use the DisplayFilterButton property to show the filter drop down button in the column's caption. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter. Use the ShowFilter method to show programmatically the column's drop down filter window. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered. The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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.

property Tree.ASCIILower as String

Specifies the set of lower characters.

Type	Description
String	A string expression that indicates the set of lower 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 (incremental search). If you want to make the auto search feature case sensitive you have to use ASCIIUpper = "" . By default, the ASCIILower property is = "abcdefghijklmnopqrstuvwxyz�סשבמךחתסיגהאוחרכטוןמלפץצע"

property Tree.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 (incremental search) feature. If you want to make the auto search feature case sensitive you have to use ASCIIUpper = "" . By default, the ASCIIUpper property is = "ABCDEFGHIJKLMNOPQRSTUVWXYZÜÉÂÄŔĹÇĘËČĎÎĚÔÖŇŰŮÁÍÓÚŃ"

method Tree.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 Tree1_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.

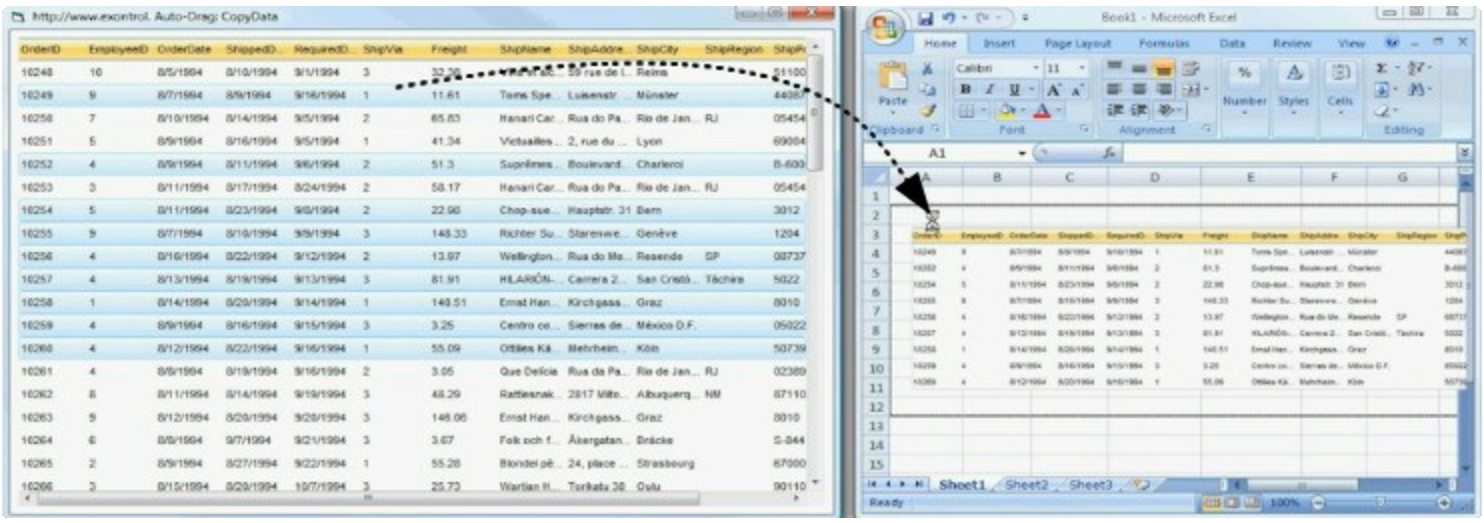
The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

property Tree.AutoDrag as AutoDragEnum

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

Type	Description
AutoDragEnum	An AutoDragEnum expression that specifies what the control does once the user clicks and start dragging an item.

By default, the AutoDrag property is exAutoDragNone(0). The AutoDrag feature indicates what the control does when the user clicks an item and starts dragging it. For instance, using the AutoDrag feature you can automatically lets the user to drag and drop the data to OLE compliant applications like Microsoft Word, Excel and so on. The SingleSel property specifies whether the control supports single or multiple selection. The AutoDrag feature adds automatically Drag and Drop, but you can still use the OLEDropMode property to handle the OLE Drag and Drop event for your custom action.



The drag and drop operation starts:

- once the user clicks and moves the cursor up or down, if the SingleSel property is True.
- once the user clicks, and waits for a short period of time, if SingleSel property is False (multiple items in selection is allowed). In this case, you can drag and drop any item that is not selected, or a contiguously selection

Once the drag and drop operation starts the mouse pointer is changed to MOVE cursor if the operation is possible, else if the Drag and Drop operation fails or if it is not possible, the mouse pointer is changed to NO cursor.

If using the AutoDrag property on:

- exAutoDragPosition

- 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 Tree.AutoSearch as Boolean

Enables or disables the auto search feature.

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 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 ExpandOnSearch property to expand items while user types characters in the control. Use the MarkSearchColumn property to specify whether the control draws a rectangle around the searching column.

The control highlights the characters as the user types them:

property Tree.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 ExTree ActiveX Control can group the columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. To change the background color of the control's locked area use BackColorLock property. Use the SelBackColor property to specify the background color for selected items. Use the CellBackColor property to assign a different background color for a specified cell. Use the ItemBackColor property to specify the item's background color. Use the BackColorAlternate property to specify the background color used to display alternate items in the control. Use the Picture property to assign a picture to the control's background.

property Tree.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.

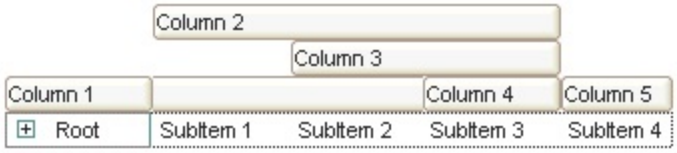
By default, the control's BackColorAlternate property is zero. The control ignores the BackColorAlternate property if it is 0 (zero). Use the BackColor property to specify the control's background color. Use the SelBackColor property to specify the selection background color.

property Tree.BackColorHeader as Color

Specifies the header's background color.

Type	Description
Color	A color expression that indicates the background color for the control's header. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the BackColorHeader and ForeColorHeader properties to customize the control's header. 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 HeaderVisible property to hide the control's header. Use the HeaderHeight property to specify the height of the control's header bar. Use the BackColor property to specify the control's background color. Use the BackColorLevelHeader property to specify the background color of the header when it displays multiple levels. Use the BackColorSortBar property to specify the background color of the control's sort bar.



The following VB sample changes the visual appearance for the control's header. Shortly, we need to add a skin to the Appearance object using the Add method, and we need to set the last 7 bits in the BackColorHeader property to indicates the index of the skin that we want to use. The sample applies the " " to the control' header bar:

```
With Tree1
   With .VisualAppearance
      .Add &H24, App.Path + "\header.ebn"
   End With
   .BackColorLevelHeader = RGB(255, 255, 255)
   .BackColorHeader = &H24000000
End With
```

The following C++ sample changes the visual aspect of the control' header bar:

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x24,
COleVariant(_T("D:\\Temp\\ExTree.Help\\header.ebn")) );
m_tree.SetBackColorHeader( 0x24000000 );
```

The following VB.NET sample changes the visual aspect of the control' header bar:

```
With AxTree1
   With .VisualAppearance
      .Add(&H24, "D:\Temp\ExTree.Help\header.ebn")
   End With
   .Template = "BackColorHeader = 603979776"
End With
```

The 603979776 value indicates the &H24000000 in hexadecimal.

The following C# sample changes the visual aspect of the control' header bar:

```
axTree1.VisualAppearance.Add(0x24, "D:\\Temp\\ExTree.Help\\header.ebn");
axTree1.Template = "BackColorHeader = 603979776";
```

The 603979776 value indicates the 0x24000000 in hexadecimal.

The following VFP sample changes the visual aspect of the control' header bar:

```
With thisform.Tree1
   With .VisualAppearance
      .Add(36, "D:\Temp\ExTree.Help\header.ebn")
   EndWith
   .BackColorHeader = 603979776
EndWith
```

property Tree.BackColorLevelHeader as Color

Specifies the multiple levels header's background color.

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

Use the BackColorHeader and ForeColorHeader properties to define colors used to paint the control's header bar. Use the BackColorLevelHeader property to specify the background color of the control's header bar when multiple levels are displayed. Use the LevelKey property to display the control's header bar using multiple levels. If the control displays the header bar using multiple levels the HeaderHeight property gets the height in pixels of a single level in the header bar. The control's header displays multiple levels if there are two or more neighbor columns with the same non empty level key.

property Tree.BackColorLock as Color

Retrieves or sets a value that indicates the control's background color for the locked area.

Type	Description
Color	A boolean expression that indicates the control's background color for the locked area.

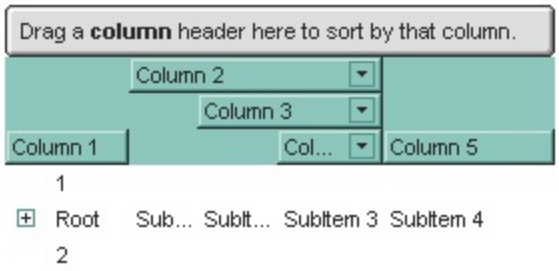
The ExTree ActiveX Control can group the columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. To change the background color of the control's unlocked area use BackColor property

property Tree.BackColorSortBar as Color

Retrieves or sets a value that indicates the sort bar's background color.

Type	Description
Color	A color expression that indicates the background color of the sort bar. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the BackColorSortBar property to specify the background color of the control's sort bar. Use the SortBarVisible property to show the control's sort bar. Use the BackColorSortBarCaption property to specify the background color of the caption of the sort bar. The caption of the sort bar is visible, if there are no columns in the sort bar. Use the SortBarCaption property to specify the caption of the sort bar. Use the ForeColorSortBar property to specify the foreground color of the control's sort bar. Use the BackColor property to specify the control's background color. Use the BackColorHeader property to specify the background color of the control's header bar. Use the BackColorLevelHeader property to specify the background color of the control's header bar when multiple levels are displayed.



The following VB sample changes the appearance for the control's sort bar. The sample uses the "" skin.

```
With Tree1
   .SortBarVisible = True
   With .VisualAppearance
      .Add &H60, App.Path + "\sortbar.ebn"
   End With
   .ForeColorSortBar = 0
   .BackColorSortBar = &H60000000
```

```
    .BackColorSortBarCaption = .BackColorSortBar
End With
```

The following C++ sample changes the appearance for the control's sort bar:

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x60,
COleVariant(_T("D:\\Temp\\ExTree.Help\\sortbar.ebn")) );
m_tree.SetSortBarVisible( TRUE );
m_tree.SetBackColorSortBar( 0x60000000 );
m_tree.SetBackColorSortBarCaption( m_tree.GetBackColorSortBar() );
```

The following VB.NET sample changes the appearance for the control's sort bar:

```
With AxTree1
    .SortBarVisible = True
    With .VisualAppearance
        .Add(&H60, "D:\Temp\ExTree.Help\sortbar.ebn")
    End With
    .Template = "BackColorSortBar = 1610612736"
    .Template = "BackColorSortBarCaption = 1610612736"
    .ForeColorSortBar = Color.Black
End With
```

The following C# sample changes the appearance for the control's sort bar:

```
axTree1.VisualAppearance.Add(0x60, "D:\\Temp\\ExTree.Help\\sortbar.ebn");
axTree1.SortBarVisible = true;
axTree1.Template = "BackColorSortBar = 1610612736";
axTree1.Template = "BackColorSortBarCaption = 1610612736";
axTree1.ForeColorSortBar = Color.Black;
```

The following VFP sample changes the appearance for the control's sort bar

```
With thisform.Tree1
    With .VisualAppearance
        .Add(96, "D:\Temp\ExTree.Help\sortbar.ebn")
    EndWith
    .SortBarVisible =.t.
```

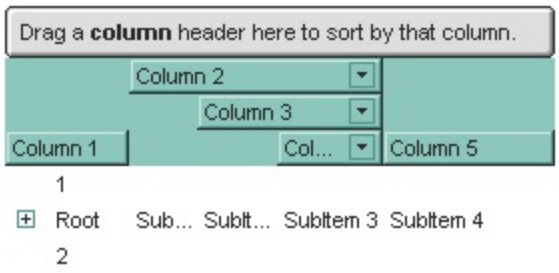
```
    .BackColorSortBar = 1610612736
    .BackColorSortBarCaption = .BackColorSortBar
    .ForeColorSortBar = 0
EndWith
```

property Tree.BackColorSortBarCaption as Color

Returns or sets a value that indicates the caption's background color in the control's sort bar.

Type	Description
Color	A color expression that indicates the caption's background color in the control's sort bar. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the SortBarCaption property to specify the caption of the sort bar, when the control's sort bar contains no columns. Use the BackColorSortBar property to specify the background color of the control's sort bar. Use the ForeColorSortBar property to specify the foreground color of the caption in the control's sort bar.



The following VB sample changes the appearance for the control's sort bar. The sample uses the "" skin.

```
With Tree1
   .SortBarVisible = True
   With .VisualAppearance
      .Add &H60, App.Path + "\sortbar.ebn"
   End With
   .ForeColorSortBar = 0
   .BackColorSortBar = &H60000000
   .BackColorSortBarCaption = .BackColorSortBar
End With
```

The following C++ sample changes the appearance for the control's sort bar:

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x60,
COleVariant(_T("D:\\Temp\\ExTree.Help\\sortbar.ebn")) );
m_tree.SetSortBarVisible( TRUE );
m_tree.SetBackColorSortBar( 0x60000000 );
m_tree.SetBackColorSortBarCaption( m_tree.GetBackColorSortBar() );
```

The following VB.NET sample changes the appearance for the control's sort bar:

```
With AxTree1
   .SortBarVisible = True
   With .VisualAppearance
      .Add(&H60, "D:\Temp\ExTree.Help\sortbar.ebn")
   End With
   .Template = "BackColorSortBar = 1610612736"
   .Template = "BackColorSortBarCaption = 1610612736"
   .ForeColorSortBar = Color.Black
End With
```

The following C# sample changes the appearance for the control's sort bar:

```
axTree1.VisualAppearance.Add(0x60, "D:\\Temp\\ExTree.Help\\sortbar.ebn");
axTree1.SortBarVisible = true;
axTree1.Template = "BackColorSortBar = 1610612736";
axTree1.Template = "BackColorSortBarCaption = 1610612736";
axTree1.ForeColorSortBar = Color.Black;
```

The following VFP sample changes the appearance for the control's sort bar

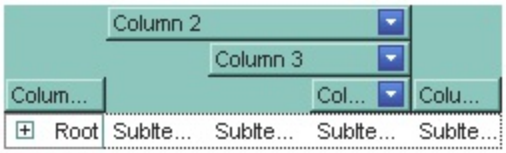
```
With thisform.Tree1
   With .VisualAppearance
      .Add(96, "D:\Temp\ExTree.Help\sortbar.ebn")
   EndWith
   .SortBarVisible =.t.
   .BackColorSortBar = 1610612736
   .BackColorSortBarCaption = .BackColorSortBar
   .ForeColorSortBar = 0
EndWith
```

property Tree.Background(Part as BackgroundPartEnum) as Color

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

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

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



The following VB sample changes the visual appearance for the "drop down" filter button. The sample applies the skin "" to the "drop down" filter buttons:

```
With Tree1
   With .VisualAppearance
      .Add &H1, App.Path + "\fbardd.ebn"
   End With
   .Background(exHeaderFilterBarButton) = &H1000000
End With
```

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

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x01,
COleVariant(_T("D:\\Temp\\ExTree.Help\\fbardd.ebn")) );
```

```
m_tree.SetBackground( 0 /*exHeaderFilterBarButton*/, 0x1000000 );
```

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

```
With AxTree1
   With .VisualAppearance
      .Add(&H1, "D:\Temp\ExTree.Help\fbardd.ebn")
   End With
   .set_Background(EXTREELib.BackgroundPartEnum.exHeaderFilterBarButton,
&H1000000)
End With
```

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

```
axTree1.VisualAppearance.Add(0x1, "D:\\Temp\\ExTree.Help\\fbardd.ebn");
axTree1.set_Background(EXTREELib.BackgroundPartEnum.exHeaderFilterBarButton,
0x1000000);
```

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

```
With thisform.Tree1
   With .VisualAppearance
      .Add(1, "D:\Temp\ExTree.Help\fbardd.ebn")
   EndWith
   .Object.Background(0) = 16777216
EndWith
```

The 16777216 value is the 0x1000000 value in hexadecimal.

method Tree.BeginUpdate ()

Maintains performance when items are added to the control one at a time.

Type	Description

This method prevents the control from painting until the EndUpdate method is called. The BeginUpdate and EndUpdate methods increases the speed of loading your items, by preventing painting the control when it suffers any change. Once that BeginUpdate method was called, you have to make sure that EndUpdate method will be called too.

The following VB sample prevents painting the control while adding data from a database:

```
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

Tree1.BeginUpdate
For Each f In rs.Fields
   Tree1.Columns.Add f.Name
Next
Tree1.PutItems rs.GetRows()
Tree1.EndUpdate
```

The following C++ sample prevents refreshing the control while adding columns and items from an ADODB recordset:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"

#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;

_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
{
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
```

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_tree.BeginUpdate();
         m_tree.SetColumnAutoResize( FALSE );
         CColumns columns = m_tree.GetColumns();
         long nCount = spRecordset->Fields->Count;
         if ( nCount > 0 )
         {
            // Adds the columns
            for ( long i = 0 ; i < nCount; i++ )
               columns.Add( spRecordset->Fields->Item[ i ]->Name );
            CItems items = m_tree.GetItems();
            // Adds the items
            while ( !spRecordset->adoEOF )
            {
               long j = 0;
               _variant_t vtI( items.AddItem( spRecordset->Fields->Item[ j ]->Value ) );
               for ( ++j ; j < nCount; j++ )
                  items.SetCellCaption( vtI, _variant_t( j ), spRecordset->Fields->Item[ j ]-
>Value );
               spRecordset->MoveNext();
            }
         }
         m_tree.EndUpdate();
      }
   }
   catch ( _com_error& e )
   {
      AfxMessageBox( e.Description() );
   }
```

```
}
```

The sample adds a column for each field in the recordset, and add a new items for each record. You can use the DataSource property to bind a recordset to the control. The #import statement imports definitions for ADODB type library, that's used to fill the control.

The following VB.NET sample prevents refreshing the control while adding columns and items:

```
With AxTree1
   .BeginUpdate()
   With .Columns
      .Add("Column 1")
      .Add("Column 2")
   End With
   With .Items
      Dim iNewItem As Integer
      iNewItem = .AddItem("Item 1")
      .CellCaption(iNewItem, 1) = "SubItem 1"
      iNewItem = .AddItem("Item 2")
      .CellCaption(iNewItem, 1) = "SubItem 2"
   End With
   .EndUpdate()
End With
```

The following C# sample prevents refreshing the control while adding columns and items:

```
axTree1.BeginUpdate();
EXTREELib.Columns columns =axTree1.Columns;
columns.Add("Column 1");
columns.Add("Column 2");
EXTREELib.Items items = axTree1.Items;
int iNewItem = items.AddItem( "Item 1" );
items.set_CellCaption( iNewItem, 1, "SubItem 1" );
items.InsertItem( iNewItem, "", "Child 1" );
iNewItem = items.AddItem( "Item 2" );
items.set_CellCaption( iNewItem, 1, "SubItem 2" );
axTree1.EndUpdate();
```

The following VFP sample prevents refreshing the control while adding new columns and items:

```
thisform.Tree1.BeginUpdate()
with thisform.Tree1.Columns
   .Add("Column 1")
   .Add("Column 2")
endwith

with thisform.Tree1.Items
   .DefaultItem = .AddItem("Item 1")
   .CellCaption(0, 1) = "SubItem 1"
   .DefaultItem = .InsertItem(.DefaultItem,"","Child 1")
   .CellCaption(0, 1) = "SubChild 1"
endwith
thisform.Tree1.EndUpdate()
```

property Tree.BorderStyle as Long

Retrieves or sets the border style of the control.

Type	Description
Long	A long expression that indicates the border style of the control.

Use the BorderStyle property to hide the control's border.

property Tree.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 RadioImage properties to define icons for radio and check box cells. The CheckImage property defines the index of the icon being used by check boxes. Use the CellHasCheckBox property to assign a checkbox to a cell. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellImage or CellImages property to assign one or multiple icons to a cell. Use the CellPicture property to assign a picture to a cell. Use the CellStateChanged event to notify your application when the cell's state is changed. Use the PartialCheck property to allow partial check feature within the column. The ImageSize property defines the size (width/height) of the control's check boxes.

method Tree.ClearFilter ()

Clears the filter.

Type	Description

The method clears the Filter and FilterType properties for all columns in the control, excepts for exNumeric and exCheck values where only the Filter property is set on empty. The ApplyFilter method is automatically called when ClearFilter method is invoked. Use the FilterBarHeight property to hide the control's filter bar. Use the FilterBarCaption property to specify the caption in the control's filter bar. Use the Description property to change predefined strings in the control's filter bar. Use the FilterCriteria property to specify the filter criteria using OR, AND or NOT operators. Use the ShowFilter method to show programmatically the column's drop down filter window.

property Tree.ColumnAutoResize as Boolean

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

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

Use the ColumnAutoResize property to fit all your columns in the client area. Use the Width property to specify the column's width. Use the SortBarColumnWidth property to specify the column's head in the control's sort bar. By default, the ColumnAutoResize property is True.

property Tree.ColumnFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as Long

Retrieves the column from point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Long	A long expression that indicates the column's index, or -1 if there is no column at the point. The property gets a negative value less or equal with 256, if the point is in the area between columns where the user can resize the column.

Use the ColumnFromPoint property to access the column from the point specified by the {X,Y} coordinates. The ColumnFromPoint property gets the index of the column when the cursor hovers the control's header bar. The X and Y coordinates are expressed in client coordinates, so a conversion must be done in case your coordinates are relative to the screen or to other window. **If the X parameter is -1 and Y parameter is -1 the ColumnFromPoint property determines the index of the column from the cursor.** Use the ItemFromPoint property to retrieve the item from cursor. The control fires the ColumnClick event when user clicks a column. Use the SortOnClick property to specify the operation that control odes when user clicks the control's header.

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

```
Private Sub Tree1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With Tree1
    Dim c As Long
    c = .ColumnFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    If (c >= 0) Then
      With .Columns(c)
        Debug.Print .Caption
      End With
    End If
  End With
End Sub
```

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

```
#include "Columns.h"
#include "Column.h"
void OnMouseMoveTree1(short Button, short Shift, long X, long Y)
{
   long nColIndex = m_tree.GetColumnFromPoint( X, Y );
   if ( nColIndex >= 0 )
   {
      CColumn column = m_tree.GetColumns().GetItem( COleVariant( nColIndex ) );
      OutputDebugString( column.GetCaption() );
   }
}
```

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

```
Private Sub AxTree1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseMoveEvent) Handles AxTree1.MouseMoveEvent
   With AxTree1
      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 axTree1_MouseMoveEvent(object sender,
AxEXTREELib._ITreeEvents_MouseMoveEvent e)
{
   int i = axTree1.get_ColumnFromPoint( e.x,e.y );
   if ( i >= 0 )
      System.Diagnostics.Debug.WriteLine( axTree1.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.Tree1
   i = .ColumnFromPoint( x, y )
   if ( i >= 0 )
      wait window nowait .Columns(i).Caption
   endif
endwith
```

property Tree.Columns as Columns

Retrieves the control's column collection.

Type	Description
Columns	A Columns object that holds the control's columns collection.

Use the Columns property to access the Columns collection. Use the Columns collection to add, remove or change columns. Use the Add method to add a new column to the control. Use the Items property to access the control's items collection. Use the AddItem, InsertItem, InsertControlItem or PutItems method to add new items to the control. Use the DataSource property to add new columns and items to the control. Adding new items fails if the control has no columns.

property Tree.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 Tree.ColumnsFloatBarSortOrder as SortOrderEnum

Specifies the sorting order for the columns being shown in the control's columns floating panel.

Type	Description
SortOrderEnum	A SortOrderEnum expression that specifies how the columns in the columns floating panel are displayed.

By default, the ColumnsFloatBarSortOrder property is SortNone. Use the ColumnsFloatBarSortOrder property to sort the columns to be displayed in the columns floating panel. The ColumnsFloatBarVisible property shows or hides the columns floating panel.

property Tree.ColumnsFloatBarVisible as ColumnsFloatBarVisibleEnum

Retrieves or sets a value that indicates whether the the columns float bar is visible or hidden.

Type	Description
ColumnsFloatBarVisibleEnum	A ColumnsFloatBarVisibleEnum expression that specifies whether the columns float bar is visible or hidden

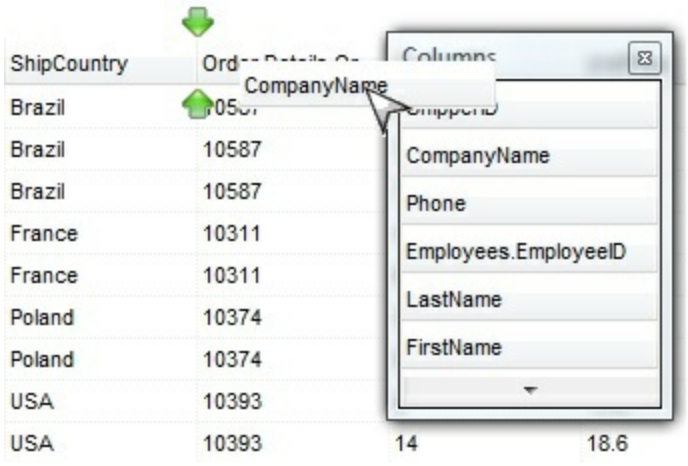
The ColumnsFloatBarVisible property indicates whether the control displays a floating panel that shows the hidden columns, so the user can drag and drop columns on order to show or hide the columns from the control. Use the ColumnsFloatBarSortOrder property to sort the columns to be displayed in the columns floating panel.

The floating panel displays the following columns:

- hidden columns, so the Visible property is False.
- drag able column, so the AllowDragging property is True.

In other words, the AllowDragging property may be used to choose if a hidden column is displayed in the floating bar. The control fires the LayoutChanged event as soon as a new column is drop on the control's header, sort or group-by bar. The Description(exColumnsFloatBar) property indicates the text to be displayed on the caption of the floating bar. The Background(exColumnsFloatAppearance) property specifies the visual appearance of the floating panel's frame.

The following screen shot shows the control's Columns float bar:



The following movies show how ColumnsFloatBarVisible works:

- The ColumnsFloatBarVisible property is used to show or hide columns by drag and drop
- The movie shows how you can customize the visual appearance of the control's

Columns floating bar

property Tree.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.

property Tree.ContinueColumnScroll as Boolean

Retrieves or sets a value indicating whether the control will automatically scroll the visible columns by pixel or by column width.

Type	Description
Boolean	A boolean expression indicating whether the control will automatically scroll the visible columns by pixel or by column width.

By default, the columns are scrolled pixel by pixel. Use the ContinueColumnScroll to scroll horizontally the control column by column. Use the EnsureVisibleColumn property to ensure that a visible column fits the control's client area. Use the Visible property to hide a column. The ScrollBySingleLine property retrieves or sets a value that indicates whether the control scrolls the lines to the end, item by item. Use the ScrollBars property to hide the control's scroll bars. Use the Scroll method to programmatically scroll the control's content.

method Tree.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 Tree1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = vbKeyC) And Shift = 2 Then
      Clipboard.Clear
      Tree1.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 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 AxTree1
  .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;
axTree1.Copy();
```

property Tree.CopyTo (File as String) as Variant

Exports the control's view to an EMF file.

Type	Description
File as String	A String expression that indicates the name of the file to be saved. If present, the CopyTo property retrieves True, if the operation succeeded, else False it is failed. If the File parameter is missing or empty, the CopyTo property retrieves an one dimension safe array of bytes that contains the EMF content. If the File parameter is not empty, the extension (characters after last dot) determines the graphical/ format of the file to be saved as follows: • ***.bmp *.dib *.rle**, saves the control's content in **BMP** format. • ***.jpg *.jpe *.jpeg *.jfif**, saves the control's content in **JPEG** format. • ***.gif**, , saves the control's content in **GIF** format. • ***.tif *.tiff**, saves the control's content in **TIFF** format. • ***.png**, saves the control's content in **PNG** format. • ***.pdf**, saves the control's content to PDF format. The File argument may carry up to 4 parameters separated by the | character in the following order: ***filename.pdf | paper size | margins | options***. In other words, you can specify the file name of the PDF document, the paper size, the margins and options to build the PDF document. By default, the paper size is 210 **mm** × 297 **mm** (A4 format) and the margins are 12.7 **mm** 12.7 **mm** 12.7 **mm** 12.7 **mm**. The units for the paper size and margins can be **pt** for PostScript Points, **mm** for Millimeters, **cm** for Centimeters, **in** for Inches and **px** for pixels. If PostScript Points are used if unit is missing. For instance, 8.27 in x 11.69 in, indicates the size of the paper in inches. Currently, the options can be **single**, which indicates that the control's content is exported to a single PDF page. For instance, the CopyTo("shot.pdf|33.11 in x 46.81 in|0 0 0 0|single") exports the control's content to an A0 single PDF page, with no margins. • ***.emf** or any other extension determines the control to

	save the control's content in **EMF** format. For instance, the CopyTo("c:\temp\snapshot.png") property saves the control's content in PNG format to snapshot.png file.
Variant	A boolean expression that indicates whether the File was successful saved, or a one dimension safe array of bytes, if the File parameter is empty string.

The CopyTo method copies/exports the control's view to BMP, PNG, JPG, GIF, TIFF, PDF or EMF graphical files, including no scroll bars. Use the Copy method to copy the control's content to the clipboard. You can use the Export method to export the control's DATA in CSV format.

- The **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- The **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- The **GIF** (Graphics Interchange Format) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- The **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.
- The **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- The **PDF** (Portable Document Format) is a file format used to present documents in a manner independent of application software, hardware, and operating systems. Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it.
- The EMF (Enhanced Metafile Format) is a 32-bit format that can contain both vector information and bitmap information. This format is an improvement over the Windows Metafile Format and contains extended features, such as the following

  Built-in scaling information
  Built-in descriptions that are saved with the file
  Improvements in color palettes and device independence

The EMF format is an extensible format, which means that a programmer can modify the original specification to add functionality or to meet specific needs. You can paste this format to Microsoft Word, Excel, Front Page, Microsoft Image Composer and any application that know to handle EMF formats.

The following VB sample saves the control's content to a file:

```
If (Tree1.CopyTo("c:\temp\test.emf")) Then
   MsgBox "test.emf file created, open it using the mspaint editor."
End If
```

The following VB sample prints the EMF content (as bytes, File parameter is empty string):

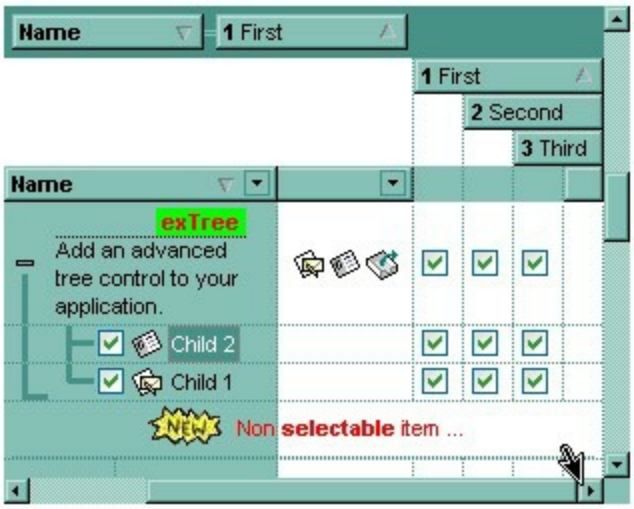
```
Dim i As Variant
For Each i In Tree1.CopyTo("")
   Debug.Print i
Next
```

property Tree.CountLockedColumns as Long

Retrieves or sets a value indicating the number of locked columns. A locked column is not scrollable.

Type	Description
Long	A long expression indicating the number of locked columns.

The ExTree ActiveX Control can group the columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. Use the BackColorLock property to change the control's background color for the locked area. Use the LockedItemCount property to add or remove items locked (fixed) to the top or bottom side of the control.



property Tree.DataSource as Object

Retrieves or sets a value that indicates the data source for object.

Type	Description
Object	An ADO or DAO Recordset object used to fill data from.

The **/COM** version provides ADO, ADODB and DAO database support. The DataSource property takes a recordset and add a column for each field found, and add a new item for each record in the recordset. Use the Visible property to hide a column. Use the 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. Use the DefaultItemHeight property before setting a DataSource property to specify the

The **/NET** version provides the following methods for data binding:

- *DataSource*, gets or sets the data source that the control is displaying data for. By default, this property is empty object. The DataSource property can be: DataTable, DataView, DataSet, DataViewManager, any component that implements the IListSource interface, or any component that implements the IList interface.
- *DataMember*, indicates a sub-list of the DataSource to show in the control. By default, this property is "". For instance, if DataSource property is a DataSet, the DataMember should indicates the name of the table to be loaded.

Click here to watch a movie on how to assign a data source to the control, in design mode, for /NET assembly.

The following VB sample binds an ADO recordset to the ExTree 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 Tree1.DataSource = rs
```

The DataSource clears the columns collection, and fill the record set into the tree, 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_tree.BeginUpdate();
         m_tree.SetColumnAutoResize( FALSE );
         m_tree.SetDataSource( spRecordset );
         m_tree.EndUpdate();
      }
   }
   catch ( _com_error& e )
   {
      AfxMessageBox( e.Description() );
   }
}
```

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

property Tree.DefaultItemHeight as Long

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

Type	Description
Long	A long expression indicates the default item height.

The DefaultItemHeight property specifies the height of the items. Changing the property fails if the control contains already items. You can change the DefaultItemHeight property at design time, or at runtime, before adding any new items to the Items collection. Use the ItemHeight property to specify the height of a specified item. Use the ScrollBySingleLine property when using the items with different heights. Use the CellSingleLine property to specify whether the cell displays the caption using multiple lines.

property Tree.Description(Type as DescriptionTypeEnum) as String

Changes descriptions for control objects.

Type	Description
Type as DescriptionTypeEnum	A DescriptionTypeEnum expression that indicates 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.

property Tree.DetectAddNew as Boolean

Specifies whether the control detects when a new record is added to the bounded record set.

Type	Description
Boolean	A boolean expression that indicates whether the control detects when a new record is added to the bounded recordset

The DetectAddNew property detects adding new records to a recordset. Use the DataSource property to bound the control to a table. If the DetectAddNew property is True, and user adds a new record to the bounded recordset, the control automatically adds a new item to the control. The DetectAddNew property has effect only if the control is bounded to an ADO, ADODB recordset, using the DataSource property.

property Tree.DrawGridLines as GridLinesEnum

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

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

Use the DrawGridLines property to add grid lines to the current view. Use the GridLineColor property to specify the color for 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.

property Tree.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 Tree.EndUpdate ()

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

Type	Description

The BeginUpdate and EndUpdate methods increases the speed of loading your items, by preventing painting the control when it suffers any change. Once that BeginUpdate method was called, you have to make sure that EndUpdate method will be called too.

The following VB sample prevents painting the control while adding data from a database:

```
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

Tree1.BeginUpdate
For Each f In rs.Fields
   Tree1.Columns.Add f.Name
Next
Tree1.PutItems rs.GetRows()
Tree1.EndUpdate
```

The following VC sample prevents refreshing the control while adding columns and items from an ADODB recordset:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"

#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;

_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
{
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
```

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_tree.BeginUpdate();
         m_tree.SetColumnAutoResize( FALSE );
         CColumns columns = m_tree.GetColumns();
         long nCount = spRecordset->Fields->Count;
         if ( nCount > 0 )
         {
            // Adds the columns
            for ( long i = 0 ; i < nCount; i++ )
               columns.Add( spRecordset->Fields->Item[ i ]->Name );
            CItems items = m_tree.GetItems();
            // Adds the items
            while ( !spRecordset->adoEOF )
            {
               long j = 0;
               _variant_t vtI( items.AddItem( spRecordset->Fields->Item[ j ]->Value ) );
               for ( ++j ; j < nCount; j++ )
                  items.SetCellCaption( vtI, _variant_t( j ), spRecordset->Fields->Item[ j ]-
>Value );
               spRecordset->MoveNext();
            }
         }
         m_tree.EndUpdate();
      }
   }
   catch ( _com_error& e )
   {
      AfxMessageBox( e.Description() );
   }
```

```
}
```

The sample adds a column for each field in the recordset, and add a new items for each record. You can use the DataSource property to bind a recordset to the control. The #import statement imports definitions for ADODB type library, that's used to fill the control.

The following VB.NET sample prevents refreshing the control while adding columns and items:

```
With AxTree1
   .BeginUpdate()
   With .Columns
      .Add("Column 1")
      .Add("Column 2")
   End With
   With .Items
      Dim iNewItem As Integer
      iNewItem = .AddItem("Item 1")
      .CellCaption(iNewItem, 1) = "SubItem 1"
      iNewItem = .AddItem("Item 2")
      .CellCaption(iNewItem, 1) = "SubItem 2"
   End With
   .EndUpdate()
End With
```

The following C# sample prevents refreshing the control while adding columns and items:

```
axTree1.BeginUpdate();
EXTREELib.Columns columns =axTree1.Columns;
columns.Add("Column 1");
columns.Add("Column 2");
EXTREELib.Items items = axTree1.Items;
int iNewItem = items.AddItem( "Item 1" );
items.set_CellCaption( iNewItem, 1, "SubItem 1" );
items.InsertItem( iNewItem, "", "Child 1" );
iNewItem = items.AddItem( "Item 2" );
items.set_CellCaption( iNewItem, 1, "SubItem 2" );
axTree1.EndUpdate();
```

The following VFP sample prevents refreshing the control while adding new columns and items:

```
thisform.Tree1.BeginUpdate()
with thisform.Tree1.Columns
   .Add("Column 1")
   .Add("Column 2")
endwith

with thisform.Tree1.Items
   .DefaultItem = .AddItem("Item 1")
   .CellCaption(0, 1) = "SubItem 1"
   .DefaultItem = .InsertItem(.DefaultItem,"","Child 1")
   .CellCaption(0, 1) = "SubChild 1"
endwith
thisform.Tree1.EndUpdate()
```

property Tree.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.

The following VB sample prevents scrolling the items, when the user calls the SortChildren method:

```
Private Sub Tree1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
   With Tree1
      Dim bEnsureOnSort As Boolean
      bEnsureOnSort = .EnsureOnSort
      .EnsureOnSort = False
      .Items.SortChildren Item, 0, False
      .EnsureOnSort = bEnsureOnSort
   End With
End Sub
```

The sample sorts the child items, when the user expands an item.

method Tree.EnsureVisibleColumn (Column as Variant)

Scrolls the control's content to ensure that the column fits the client area.

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

The EnsureVisibleColumn method ensures that the given column fits the control's client area. The EnsureVisibleColumn method has no effect if the column is hidded. Use the Visible property to show or hide a column. Use the Position property to change the column's position. Use the EnsureVisibleItem method to ensure that an item fits the control's client area. Use the ScrollBars property to hide the control's scroll bars. Use the Scroll method to programmatically scroll the control's content.

property Tree.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 Tree.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed

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

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

For instance, the following sample retrieves the control's background color:

```
Debug.Print Tree1.ExecuteTemplate("BackColor")
```

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

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

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

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

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier*

property Tree.ExpandOnDblClick as Boolean

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

Type	Description
Boolean	A boolean expression that indicates whether an item is expanded on dbl click.

Use the ExpandOnDblClick property to disable expanding or collapsing items when user dbl clicks an item. By default, the ExpandOnDblClick property is True. Use the ExpandOnKeys property to specify whether the control expands or collapses a node when user presses arrow keys. 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. The control fires the DblClick event when user double clicks the control. Use the ExpandItem property to programmatically expand or collapse an item.

property Tree.ExpandOnKeys as Boolean

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

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

Use the ExpandOnKeys property to specify whether the control expands or collapses a node when user presses arrow keys. By default, the ExpandOnKeys property is True. Use the ExpandOnDblClick property to specify whether the control expands or collapses a node when user dbl clicks a node. The ExpandOnSearch property specifies whether the control expands nodes when incremental searching is on (AutoSearch property is different than 0) and user types characters when the control has the focus. If the ExpandOnKeys property is False, the user can't expand or collapse the items using the + or - keys on the numeric keypad. Use the ExpandItem property to programmatically expand or collapse an item.

The following VB sample expands or collapses the focused item if the user presses the + or - keys on the numeric keypad, and ExpandOnKeys property is False:

```
Private Sub Tree1_KeyDown(KeyCode As Integer, Shift As Integer)
  With Tree1.Items
    If (KeyCode = vbKeyAdd) Then
      .ExpandItem(.FocusItem) = True
    End If
    If (KeyCode = vbKeySubtract) Then
      .ExpandItem(.FocusItem) = False
    End If
  End With
End Sub
```

The following C++ sample expands or collapses the focused item if the user presses the + or - keys on the numeric keypad, and ExpandOnKeys property is False:

```
#include "Items.h"
void OnKeyDownTree1(short FAR* KeyCode, short Shift)
{
  CItems items = m_tree.GetItems();
  switch ( *KeyCode )
```

```
    {
        case VK_ADD:
        case VK_SUBTRACT:
        {
            items.SetExpandItem( items.GetFocusItem(), *KeyCode == VK_ADD ? TRUE : FALSE
);
            break;
        }
    }
}
```

The following VB.NET sample expands or collapses the focused item if the user presses the + or - keys on the numeric keypad, and ExpandOnKeys property is False:

```
Private Sub AxTree1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_KeyDownEvent) Handles AxTree1.KeyDownEvent
    Select Case (e.keyCode)
        Case Keys.Add
            With AxTree1.Items
                .ExpandItem(.FocusItem) = True
            End With
        Case Keys.Subtract
            With AxTree1.Items
                .ExpandItem(.FocusItem) = False
            End With
    End Select
End Sub
```

The following C# sample expands or collapses the focused item if the user presses the + or - keys on the numeric keypad, and ExpandOnKeys property is False:

```
private void axTree1_KeyDownEvent(object sender,
AxEXTREELib._ITreeEvents_KeyDownEvent e)
{
    if ( ( e.keyCode == Convert.ToInt16(Keys.Add)  ) || (e.keyCode ==
Convert.ToInt16(Keys.Subtract) ) )
        axTree1.Items.set_ExpandItem( axTree1.Items.FocusItem, e.keyCode ==
Convert.ToInt16(Keys.Add) );
```

```
}
```

The following VFP sample expands or collapses the focused item if the user presses the + or - keys on the numeric keypad, and ExpandOnKeys property is False:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

with thisform.Tree1.Items
   if ( keycode = 107 )
      .DefaultItem = .FocusItem
      .ExpandItem(0) = .t.
   else
      if ( keycode = 109 )
         .ExpandItem(0) = .f.
      endif
   endif
endwith
```

property Tree.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 Tree.Export ([Destination as Variant], [Options as Variant])

Exports the control's data to a CSV or HTML format.

Type	Description
Destination as Variant	A String expression that specifies the file/format to be created. The Destination parameter indicates the format to be created as follows: • "array" indicates that the Export method returns the control's data as a two-dimensional array • if "htm" or "html", the control returns the HTML format (including CSS style) • Any file-name that ends on ".htm" or ".html" creates the file with the HTML format inside • missing, empty, or any other case the Export exports the control's data in CSV format. No error occurs, if the Export method can not create the file.
Options as Variant	A String expression that specifies the options to be used when exporting the control's data, as explained bellow.

Return	Description
Variant	The result of the Export method is a: • two-dimensional array, if the Destination is "array". For instance Export("array","vis") method exports the control's data as it is displayed into a two-dimensional array (zero-based). The result includes the columns headers into the first line, while the rest of lines contains the control's visible data. For instance, Export("array", "vis")(1, 5) returns the value of the cell on the second column and fifth row. • string, that indicates the format being exported. It could be CSV or HTML format based on the Destination parameter

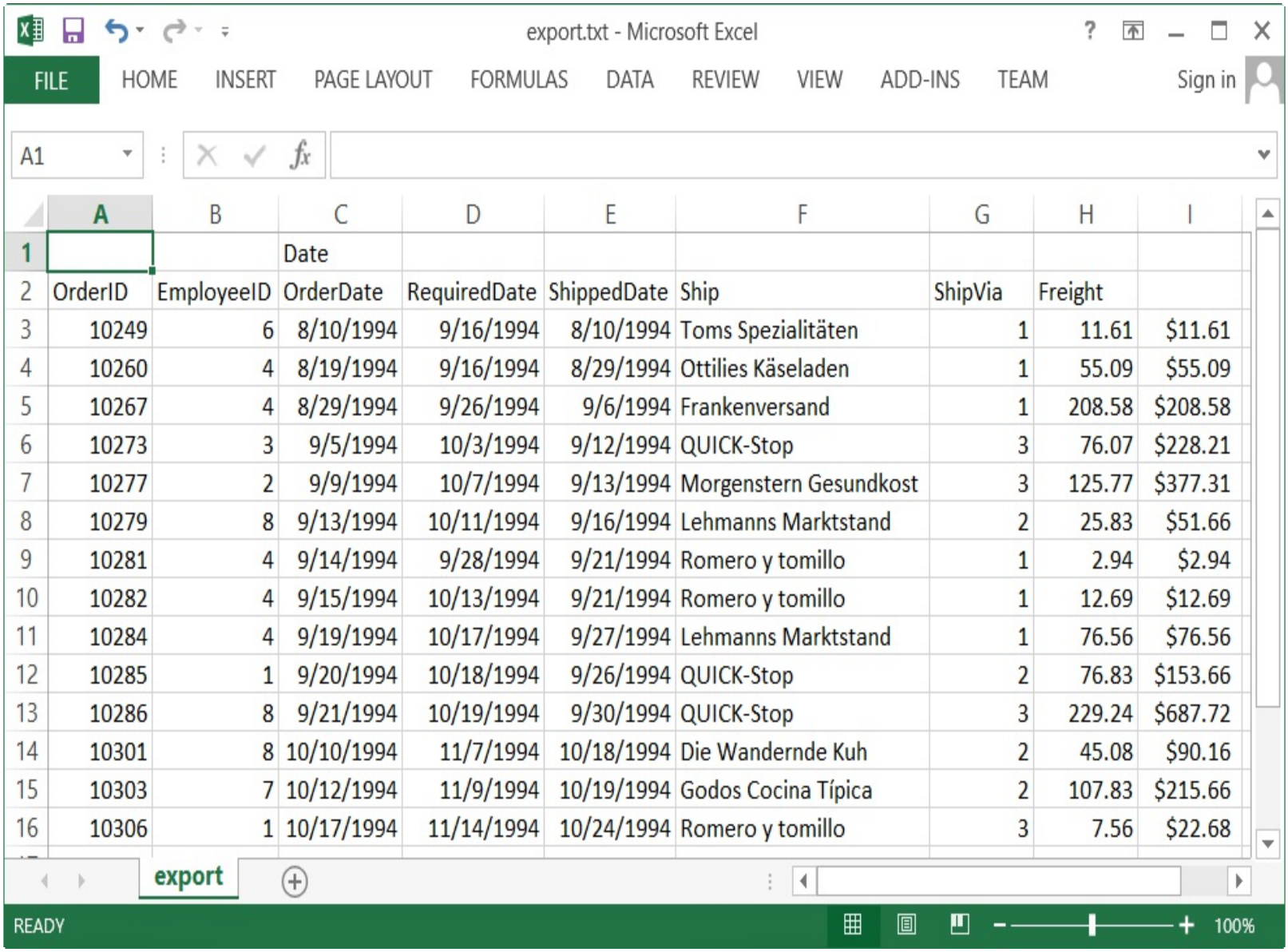
The Export method can export the control's DATA to a CSV or HTML format. The Export method can export a collection of columns from selected, visible, check or all items. By default, the control export all items, unless there is no filter applied on the control, where only visible items are exported. No visual appearance is saved in CSV format, instead the

HTML format includes the visual appearance in CSS style.

Let's say we have the following control's DATA:

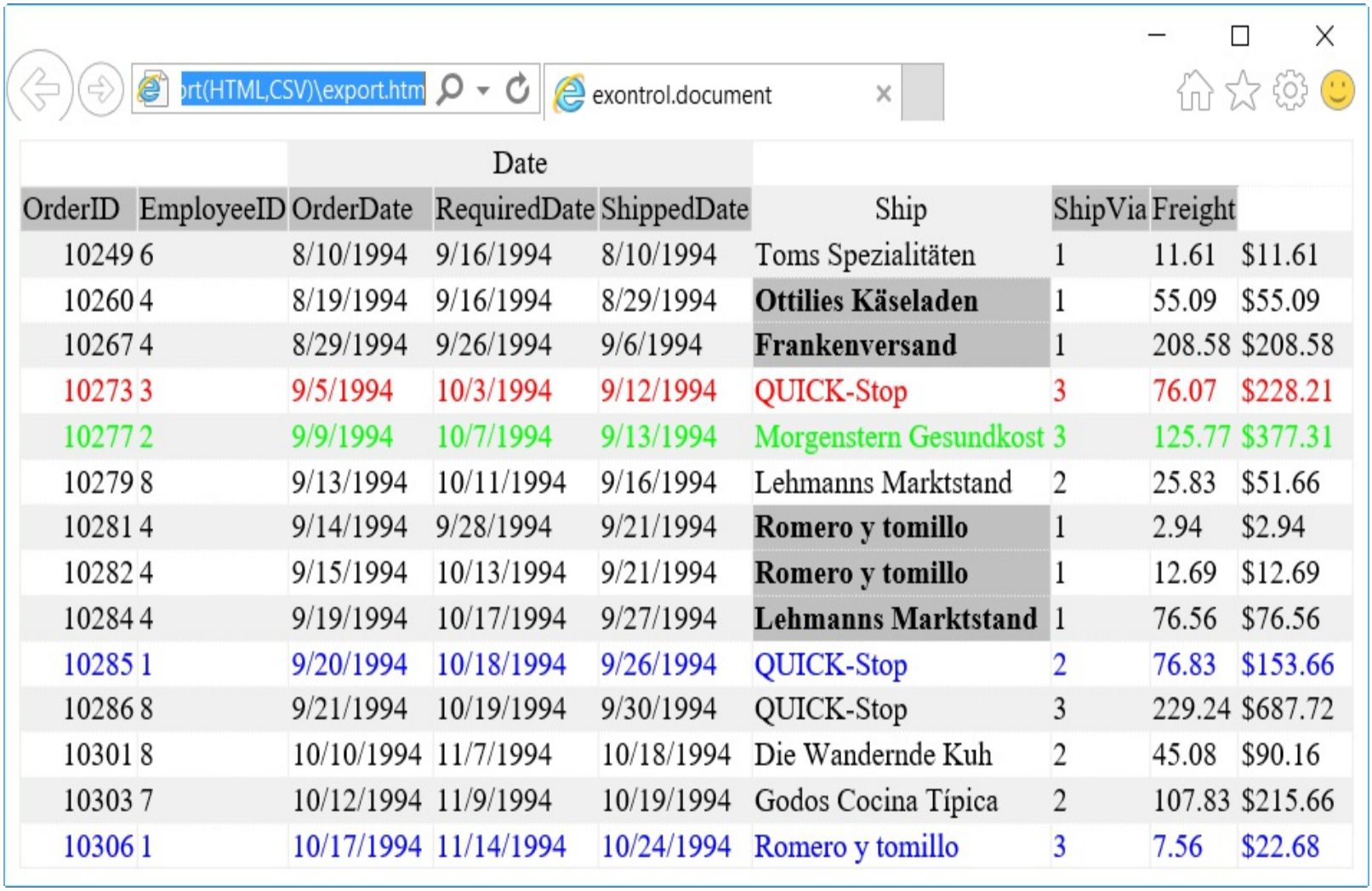
		Date						
OrderID	Empl...	OrderDate	RequiredDate	ShippedDate	Ship	ShipVia	Freight	Total
10249	6	8/10/1994	9/16/1994	8/10/1994	Toms Spezialitäten	1	11.61	$11.61
10260	4	8/19/1994	9/16/1994	8/29/1994	Ottilies Käseladen	1	55.09	$55.09
10267	4	8/29/1994	9/26/1994	9/6/1994	Frankenversand	1	208.58	$208.58
10273	3	9/5/1994	10/3/1994	9/12/1994	QUICK-Stop	3	76.07	$228.21
10277	2	9/9/1994	10/7/1994	9/13/1994	Morgenstern Gesundkost	3	125.77	$377.31
10279	8	9/13/1994	10/11/1994	9/16/1994	Lehmanns Marktstand	2	25.83	$51.66
10281	4	9/14/1994	9/28/1994	9/21/1994	Romero y tomillo	1	2.94	$2.94
10282	4	9/15/1994	10/13/1994	9/21/1994	Romero y tomillo	1	12.69	$12.69
10284	4	9/19/1994	10/17/1994	9/27/1994	Lehmanns Marktstand	1	76.56	$76.56
10285	1	9/20/1994	10/18/1994	9/26/1994	QUICK-Stop	2	76.83	$153.66
10286	8	9/21/1994	10/19/1994	9/30/1994	QUICK-Stop	3	229.24	$687.72
10301	8	10/10/1994	11/7/1994	10/18/1994	Die Wandernde Kuh	2	45.08	$90.16
10303	7	10/12/1994	11/9/1994	10/19/1994	Godos Cocina Típica	2	107.83	$215.66
10306	1	10/17/1994	11/14/1994	10/24/1994	Romero y tomillo	3	7.56	$22.68

The following screen shot shows the control's DATA in CSV format:



	A	B	C	D	E	F	G	H	I
1			Date						
2	OrderID	EmployeeID	OrderDate	RequiredDate	ShippedDate	Ship	ShipVia	Freight	
3	10249	6	8/10/1994	9/16/1994	8/10/1994	Toms Spezialitäten	1	11.61	$11.61
4	10260	4	8/19/1994	9/16/1994	8/29/1994	Ottilies Käseladen	1	55.09	$55.09
5	10267	4	8/29/1994	9/26/1994	9/6/1994	Frankenversand	1	208.58	$208.58
6	10273	3	9/5/1994	10/3/1994	9/12/1994	QUICK-Stop	3	76.07	$228.21
7	10277	2	9/9/1994	10/7/1994	9/13/1994	Morgenstern Gesundkost	3	125.77	$377.31
8	10279	8	9/13/1994	10/11/1994	9/16/1994	Lehmanns Marktstand	2	25.83	$51.66
9	10281	4	9/14/1994	9/28/1994	9/21/1994	Romero y tomillo	1	2.94	$2.94
10	10282	4	9/15/1994	10/13/1994	9/21/1994	Romero y tomillo	1	12.69	$12.69
11	10284	4	9/19/1994	10/17/1994	9/27/1994	Lehmanns Marktstand	1	76.56	$76.56
12	10285	1	9/20/1994	10/18/1994	9/26/1994	QUICK-Stop	2	76.83	$153.66
13	10286	8	9/21/1994	10/19/1994	9/30/1994	QUICK-Stop	3	229.24	$687.72
14	10301	8	10/10/1994	11/7/1994	10/18/1994	Die Wandernde Kuh	2	45.08	$90.16
15	10303	7	10/12/1994	11/9/1994	10/19/1994	Godos Cocina Típica	2	107.83	$215.66
16	10306	1	10/17/1994	11/14/1994	10/24/1994	Romero y tomillo	3	7.56	$22.68

The following screen shot shows the control's DATA in HTML format:



		Date						
OrderID	EmployeeID	OrderDate	RequiredDate	ShippedDate	Ship	ShipVia	Freight	
10249	6	8/10/1994	9/16/1994	8/10/1994	Toms Spezialitäten	1	11.61	$11.61
10260	4	8/19/1994	9/16/1994	8/29/1994	**Ottilies Käseladen**	1	55.09	$55.09
10267	4	8/29/1994	9/26/1994	9/6/1994	**Frankenversand**	1	208.58	$208.58
10273	3	9/5/1994	10/3/1994	9/12/1994	QUICK-Stop	3	76.07	$228.21
10277	2	9/9/1994	10/7/1994	9/13/1994	Morgenstern Gesundkost	3	125.77	$377.31
10279	8	9/13/1994	10/11/1994	9/16/1994	Lehmanns Marktstand	2	25.83	$51.66
10281	4	9/14/1994	9/28/1994	9/21/1994	**Romero y tomillo**	1	2.94	$2.94
10282	4	9/15/1994	10/13/1994	9/21/1994	**Romero y tomillo**	1	12.69	$12.69
10284	4	9/19/1994	10/17/1994	9/27/1994	**Lehmanns Marktstand**	1	76.56	$76.56
10285	1	9/20/1994	10/18/1994	9/26/1994	QUICK-Stop	2	76.83	$153.66
10286	8	9/21/1994	10/19/1994	9/30/1994	QUICK-Stop	3	229.24	$687.72
10301	8	10/10/1994	11/7/1994	10/18/1994	Die Wandernde Kuh	2	45.08	$90.16
10303	7	10/12/1994	11/9/1994	10/19/1994	Godos Cocina Típica	2	107.83	$215.66
10306	1	10/17/1994	11/14/1994	10/24/1994	Romero y tomillo	3	7.56	$22.68

The Options parameter consists a list of fields separated by | character, in the following order:

1. The first field could be **all**, **vis**, **sel** or **chk**, to export all, just visible, selected or checked items. The all option is used, if the field is missing. The **all** option displays all items, including the hidden or collapsed items. The **vis** option includes the visible items only, not including the child items of a collapsed item, or not-visible items (item's height is 0). The **sel** options lists the items being selected. The **chk** option lists all check and visible items. If chk option is used, the first column in the columns list should indicate the index of the column being queried for a check box state.
2. the second field indicates the column to be exported. All visible columns are exported, if missing. The list of columns is separated by , character, and indicates the index of the column to be shown on the exported data. The first column in the list indicates the column being queried, if the option **chk** is used.
3. the third field indicates the character to separate the fields inside each exported line [tab character-if missing]. This field is valid, only when exporting to a CSV format
4. the forth field could be **ansi** or **unicode**, which indicates the character-set to save the control's content to Destination. For instance, Export(Destination,"|||unicode") saves the control's content to destination in UNICODE format (two-bytes per character). By default, the Export method creates an ANSI file (one-byte character)

The Destination parameter indicates the file to be created where exported date should be saved. For instance, Export("c:\temp\export.html") exports the control's DATA to export.html file in HTML format, or Export("","sel|0,1|;") returns the cells from columns 0, 1 from the selected items, to a CSV format using the ; character as a field separator.

The "CSV" refers to any file that:

- CSV stands for Comma Separated Value
- is plain text using a character set such as ASCII, Unicode,
- consists of records (typically one record per line),
- with the records divided into fields separated by delimiters (typically a single reserved character such as tab, comma, or semicolon; sometimes the delimiter may include optional spaces),
- where every record has the same sequence of fields

The "HTML" refers to any file that:

- HTML stands for HyperText Markup Language.
- is plain text using a character set such as ASCII, Unicode
- It's the way web pages are encoded to handle things like bold, italics and even color text red.

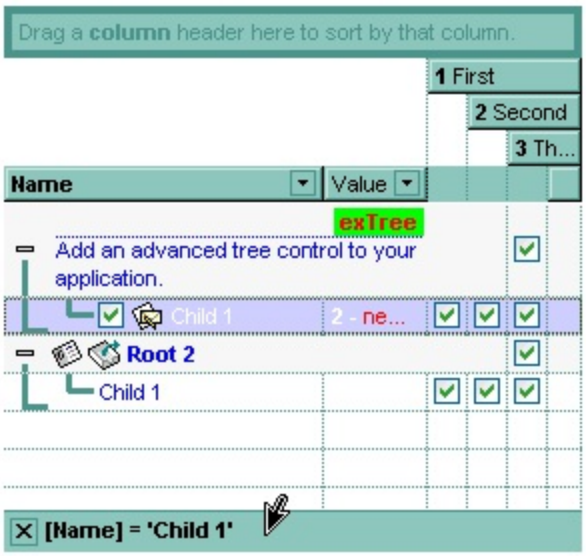
You can use the Copy/CopyTo to export the control's view to clipboard/EMF/BMP/JPG/PNG/GIF or PDF format.

property Tree.FilterBarBackColor as Color

Specifies the background color of the control's filter bar.

Type	Description
Color	A color expression that defines the background color for description of the control's filter. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

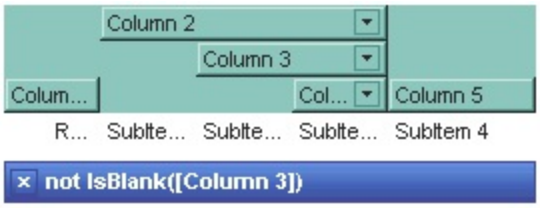
Use the FilterBarForeColor and FilterBarBackColor properties to define the colors used to paint the description for control's filter. Use the FilterBarHeight property to hide the control's filter bar header. Use the BackColor property to specify the control's background color. Use the BackColorLevelHeader property to specify the background color of the header when it displays multiple levels. Use the BackColorSortBar property to specify the background color of the control's sort bar.



The following VB sample changes the visual appearance for control's filter bar. The sample applies the skin "" to the "close" button in the control's filter bar, and the "" skin to the filter bar caption area:

```
With Tree1
  With .VisualAppearance
    .Add &H2, App.Path + "\fbarclose.ebn"
    .Add &H12, App.Path + "\filterbar.ebn"
```

```
    End With
    .Background(exFooterFilterBarButton) = &H2000000
    .FilterBarBackColor = &H12000000
    .FilterBarForeColor = RGB(255, 255, 255)
End With
```



The following C++ sample changes the visual appearance for the "close" button in the control's filter bar:

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x2,
COleVariant(_T("D:\\Temp\\ExTree.Help\\fbarclose.ebn")) );
m_tree.GetVisualAppearance().Add( 0x12,
COleVariant(_T("D:\\Temp\\ExTree.Help\\filterbar.ebn")) );
m_tree.SetBackground( 1 /*exFooterFilterBarButton*/, 0x2000000 );
m_tree.SetFilterBarBackColor( 0x12000000 );
m_tree.SetFilterBarForeColor( RGB(255,255,255) );
```

The following VB.NET sample changes the visual appearance for the "close" button in the control's filter bar:

```
With AxTree1
    With .VisualAppearance
        .Add(&H2, "D:\Temp\ExTree.Help\fbarclose.ebn")
        .Add(&H12, "D:\Temp\ExTree.Help\filterbar.ebn")
    End With
    .Template = "FilterBarBackColor = 301989888"
    .FilterBarForeColor = Color.White
    .set_Background(EXTREELib.BackgroundPartEnum.exFooterFilterBarButton, &H2000000)
End With
```

The following C# sample changes the visual appearance for the "close" button in the control's filter bar:

```
axTree1.VisualAppearance.Add(0x2, "D:\\Temp\\ExTree.Help\\fbarclose.ebn");
```

```
axTree1.VisualAppearance.Add(0x12, "D:\\Temp\\ExTree.Help\\filterbar.ebn");
axTree1.set_Background(EXTREELib.BackgroundPartEnum.exFooterFilterBarButton,
0x2000000);
axTree1.Template = "FilterBarBackColor = 301989888";
axTree1.FilterBarForeColor = Color.White;
```

The following VFP sample changes the visual appearance for the "close" button in the control's filter bar:

```
With thisform.Tree1
   With .VisualAppearance
      .Add(2, "D:\Temp\ExTree.Help\fbarclose.ebn")
      .Add(18, "D:\Temp\ExTree.Help\filterbar.ebn")
   EndWith
   .Object.Background(1) = 33554432
   .FilterBarBackColor = 301989888
   .FilterBarForeColor = RGB(255,255,255)
EndWith
```

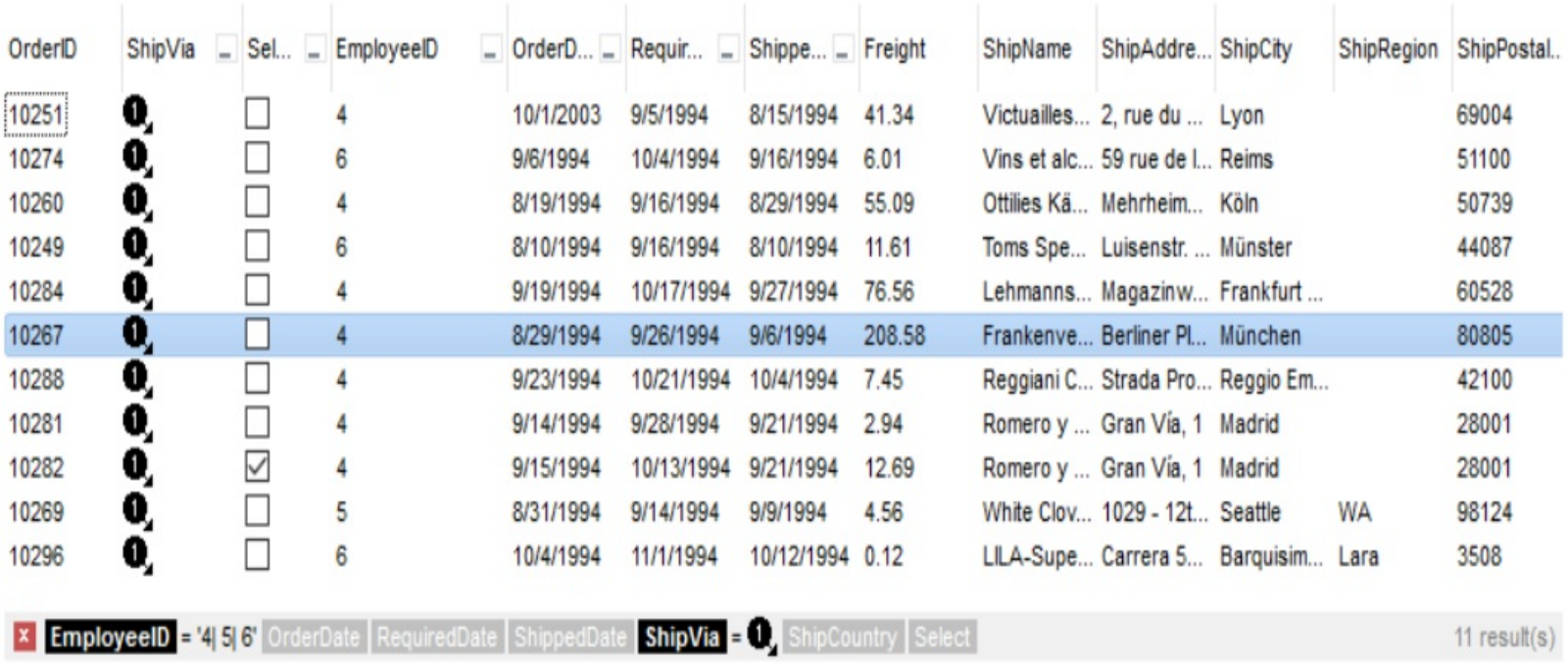
The 301989888 value is the 0x12000000 value in hexadecimal.

property Tree.FilterBarCaption as String

Specifies the filter bar's caption.

Type	Description
String	A string value that defines the expression to display the control's filter bar.

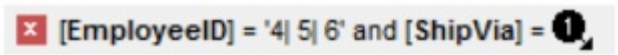
By default, the FilterBarCaption property is empty. You can use the FilterBarCaption property to define the way the filter bar's caption is being displayed. The FilterBarCaption is displayed on the bottom side of the control where the control's filter bar is shown. While the FilterBarCaption property is empty, the control automatically builds the caption to be displayed on the filter bar from all columns that participates in the filter using its name and values. For instance, if the control filters items based on the columns "EmployeeID" and "ShipVia", the control's filter bar caption would appear such as "[EmployeeID] = '...' and [ShipVia] = '...'". The FilterBarCaption property supports expressions as explained bellow.



For instance:

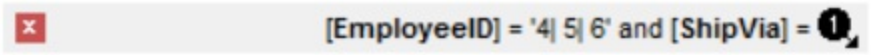
- "no filter", shows no filter caption all the time

- "" displays no filter bar, if no filter is applied, else it displays the current filter



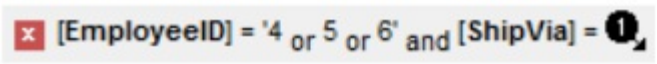
- "`<r>` + value", displays the current filter caption aligned to the right. You can include

the exFilterBarShowCloseOnRight flag into the FilterBarPromptVisible property to display the close button aligned to the right



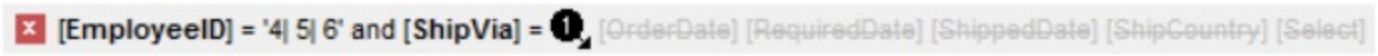
- "value replace ` and ` with `<fgcolor=FF0000> and </fgcolor>`", replace the AND keyword with a different foreground color

- "value replace ` and ` with `<off 4> and </off>` replace `|` with ` <off 4>or</off> ` replace ` ` with ` `", replaces the AND and | values

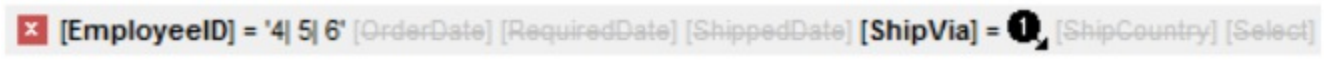


- "value replace `[` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `]` with ` </b></bgcolor></fgcolor>`", highlights the columns being filtered with a different background/foreground colors.

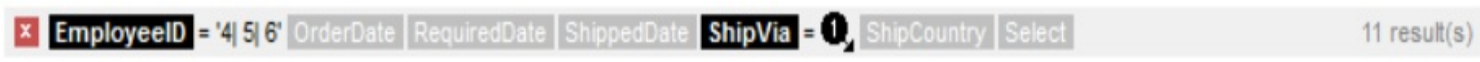
- "value + ` ` + available", displays the current filter, including all available columns to be filtered



- "allui" displays all available columns



- "((allui + `<fgcolor=808080>` + (matchitemcount < 0 ? ((len(allui) ? `` : ``) + `<r>` + abs(matchitemcount + 1) + ` result(s)`) : (`<r><fgcolor=808080>`+ itemcount + ` item(s)`))) replace `[<b>` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `</b>]` with ` </b></bgcolor></fgcolor>` replace `[<s>` with `<bgcolor=C0C0C0> <fgcolor=FFFFFF> ` replace `</s>]` with ` </bgcolor></fgcolor>`)" displays all available columns to be filtered with different background/foreground colors including the number of items/results



Use the FilterBarForeColor and FilterBarBackColor properties to define the colors used to paint the description for control's filter. Use the FilterBarHeight property to specify the

height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar. Use the Description property to define predefined strings in the filter bar caption. The VisibleItemCount property specifies the number of visible items in the list. The MatchItemCount property returns the number of matching items. The FilterBarPromptVisible property specifies whether how/where the control's filter/prompt is shown.

The FilterBarCaption method supports the following keywords, constants, operators and functions:

- **value** or **current** keyword returns the current filter as a string. At runtime the value may return a string such as "[EmployeeID] = '4| 5| 6' and [ShipVia] = <img>1</img>", so the control automatically applies HTML format, which you can change it. For instance, "upper(value)" displays the caption in uppercase or "value replace `<b>` with `<fgcolor=808080>` replace `</b>` with `</fgcolor>`" displays the column's name with a different foreground color.
- **itemcount** keyword returns the total number of items as indicated by 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 macthing items, if filter is applied, aligned to the right
- **leafitemcount** keyword returns the number of leaf items. A leaf item is an item with no

child items. At runtime, the leafitemcount is a positive number that computes the number of leaf items (expanded or collapsed). For instance, the "value + `<r><fgcolor=808080><font ;6>` + leafitemcount" displays the number of leaf items aligned to the right with a different font and foreground color.

- **promptpattern** returns the pattern in the filter bar's prompt, as a string. The FilterBarPromptPattern specifies the pattern for the filter prompt. The control's filter bar prompt is visible, if the exFilterBarPromptVisible flag is included in the FilterBarPromptVisible property.
- **available** keyword returns the list of columns that are not currently part of the control's filter, but are available to be filtered. A column is available to be filtered, if the DisplayFilterButton property of the Column object, is True. At runtime, the available keyword may return a string such as "<fgcolor=C0C0C0>[<s>OrderDate</s>]<fgcolor> </fgcolor>[<s>RequiredDate</s>]<fgcolor> </fgcolor>[<s>ShippedDate</s>]<fgcolor> </fgcolor>[<s>ShipCountry</s>]<fgcolor> </fgcolor>[<s>Select</s>]</fgcolor>", so the control automatically applies HTML format, which you can change it. For instance, "value + ` ` + available", displays the current filter, including all available columns to be filtered. For instance, the "value + `<r>` + available replace `C0C0C0` with `FF0000`" displays the available columns aligned to the right with a different foreground color.
- **allui** keyword returns the list of columns that are part of the current filter and available columns to be filtered. A column is available to be filtered, if the DisplayFilterButton property of the Column object, is True. At runtime, the allui keyword may return a string such as "[EmployeeID] = '4| 5| 6'<fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>OrderDate</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>RequiredDate</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>ShippedDate</s>]</fgcolor><fgcolor> </fgcolor>[ShipVia] = <img>1</img><fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>ShipCountry</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>Select</s>]</fgcolor>", so the control automatically applies HTML format, which you can change it. For instance, "allui", displays the current filter, including all available columns to be filtered. For instance, the "((allui + `<fgcolor=808080>` + (matchitemcount < 0 ? ((len(allui) ? `` : ``) + `<r>` + abs(matchitemcount + 1) + ` result(s)`) : (`<r><fgcolor=808080>`+ itemcount + ` item(s)`))) replace `[<b>` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `</b>]` with ` </b></bgcolor></fgcolor>` replace `[<s>` with `<bgcolor=C0C0C0><fgcolor=FFFFFF> ` replace `</s>]` with ` </bgcolor></fgcolor>`)" displays all available columns to be filtered with different background/foreground colors including the number of items/results
- **all** keyword returns the list of all columns (visible or hidden) no matter if the DisplayFilterButton property is True or False. At runtime, the all keyword may return a string such as "<fgcolor=C0C0C0>[<s>OrderID</s>]</fgcolor><fgcolor> </fgcolor>[EmployeeID] = '4| 5| 6'<fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>OrderDate</s>]</fgcolor><fgcolor> </fgcolor><fgcolor=C0C0C0>[<s>RequiredDate</s>]</fgcolor><fgcolor>", so the control automatically applies

HTML format, which you can change it. For instance, "all", displays the current filter, including all other columns. For instance, the "((all + `<fgcolor=808080>` + (matchitemcount < 0 ? ((len(allui) ? `` : ``) + `<r>` + abs(matchitemcount + 1) + ` result(s)`) : (`<r><fgcolor=808080>`+ itemcount + ` item(s)`))) replace `[<b>` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `</b>]` with ` </b></bgcolor> </fgcolor>` replace `[<s>` with `<bgcolor=C0C0C0><fgcolor=FFFFFF> ` replace `</s>]` with ` </bgcolor></fgcolor>`)" displays all columns with different background/foreground colors including the number of items/results

Also, the FilterBarCaption property supports predefined constants and operators/functions as described here.

Also, the FilterBarCaption property supports HTML format as described here:

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

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

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

  Any ex-HTML caption can be transformed to an expandable-caption, by inserting the anchor ex-HTML tag. For instance, "<solidline>Header</solidline>
Line1<r><a ;exp=show lines>+
Line2
Line3" shows the Header in

underlined and bold on the first line and Line1, Line2, Line3 on the rest. The "show lines" is shown instead of Line1, Line2, Line3 once the user clicks the + sign.

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

character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold

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

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:

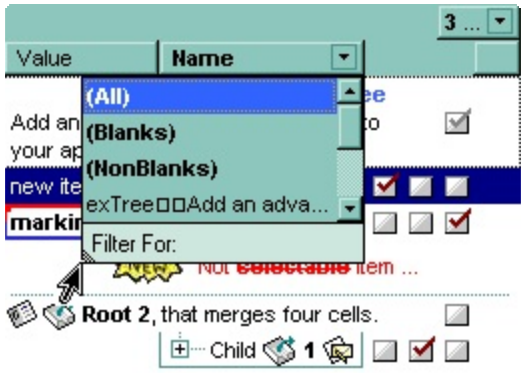

outline anti-aliasing

property Tree.FilterBarDropDownHeight as Double

Specifies the height of the drop down filter window proportionally with the height of the control's list.

Type	Description
Double	A double expression that indicates the height of the drop down filter window. The meaning of the value is explained bellow.

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 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. Use the FilterBarDropDownWidth property to specify the width of the drop down filter window. Use the Description property to define predefined strings in the filter bar. At run-time, the user can resize the drop down filter window by clicking the left-bottom ticker. The FilterBarDropDownHeight property is changed if the user resizes the drop down filter window. Use the ShowFilter method to show programmatically the column's drop down filter window. The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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.

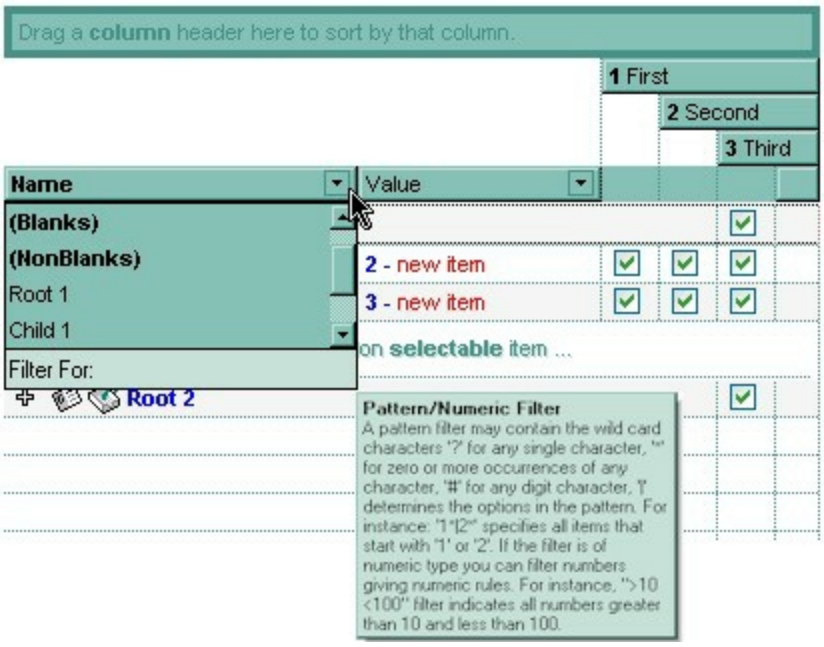


If the FilterBarDropDownHeight property is negative, the absolute value of the FilterBarDropDownHeight property indicates the height of the drop down filter window in pixels. In this case, the height of the drop down filter window is not proportionally with the height of the control's list area. For instance, the following sample specifies the height of the drop down filter window being 100 pixels:

```
With Tree1
   .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 Tree1
   .FilterBarDropDownHeight = 1
End With
```



property Tree.FilterBarFont as IFontDisp

Retrieves or sets the font for control's filter bar.

Type	Description
IFontDisp	A font object that indicates the font used to paint the description for control's filter

Use the FilterBarFont property to specify the font for the control's filter bar object. Use the Font property to set the control's font. Use the FilterBarHeight property to specify the height of the filter bar. Use the FilterBarCaption property to define the control's filter bar caption. Use the Refresh method to refresh the control.

property Tree.FilterBarForeColor as Color

Specifies the foreground color of the control's filter bar.

Type	Description
Color	A color expression that defines the foreground color of the description of the control's filter.

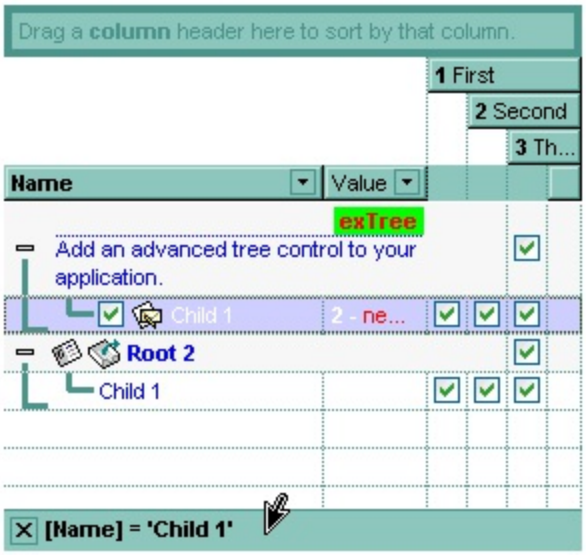
Use the FilterBarForeColor and FilterBarBackColor properties to define colors used to paint the description of the control's filter. Use the FilterBarFont property to specify the filter bar's font. Use the FilterBarCaption property to specify the caption of the control's filter bar.

property Tree.FilterBarHeight as Long

Specifies the height of the control's filter bar. If the value is less than 0, the filter bar is automatically resized to fit its description.

Type	Description
Long	A long expression that indicates the height of the filter bar 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 grater than zero it defines the height in pixels of the filter's description. Use the ClearFilter method to clear the control's filter. Use the FilterBarCaption property to define the control's filter bar caption. Use the FilterBarFont property to specify the font for the control's filter bar. Use the FilterBarDropDownWidth property to specify the width of the drop down filter window. Use the FilterBarDropDownHeight to specify the height of the drop down filter window. Use the ShowFilter method to show programmatically the column's drop down filter window.



property Tree.FilterBarPrompt as String

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

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

By default, the FilterBarPrompt property is "<fgcolor=808080>Start Filter...</fgcolor>". The FilterBarPromptPattern property specifies the pattern to filter the list using the filter prompt feature. Changing the FilterBarPrompt property won't change the current filter. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered. Use the FilterBarCaption property to change the caption in the filter bar once a new filter is applied. The FilterBarFont property specifies the font to be used in the filter bar. The FilterBarBackColor property specifies the background color or the visual aspect of the control's filter bar. The FilterBarForeColor property specifies the foreground color or the control's filter bar.

The FilterBarPrompt property supports HTML format as described here:

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

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

  - exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
  - e64, encodes in BASE64 the HTML text to be shown once the user clicks the

anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu " that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY string encodes the "<fgcolor 808080>show lines<a>-</fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part

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

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

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

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

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

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

The FilterBarPrompt property has effect only if:

- FilterBarPromptVisible property is True
- FilterBarPromptPattern property is Empty.

property Tree.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 Tree.FilterBarPromptPattern as String

Specifies the pattern for the filter prompt.

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

By default, the FilterBarPromptPattern property is empty. If the FilterBarPromptPattern property is empty, the filter bar displays the FilterBarPrompt property, if the FilterBarPromptVisible property is True. The FilterBarPromptPattern property indicates the patter to filter the list. The pattern may include wild characters if the FilterBarPromptType property is exFilterPromptPattern. The FilterBarPromptColumns specifies the list of columns to be used when filtering. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way.

The following samples shows the filter prompt, and filter for items that contains "london":

Access

```
With Tree1
   .BeginUpdate
   .ColumnAutoResize = True
   .ContinueColumnScroll = 0
   .MarkSearchColumn = False
   .SearchColumnIndex = 1
   .FilterBarPromptVisible = True
   .FilterBarPromptPattern = "london"
   With .Columns
      .Add("Name").Width = 96
      .Add("Title").Width = 96
      .Add "City"
   End With
   With .Items
      h0 = .AddItem("Nancy Davolio")
      .CellCaption(h0,1) = "Sales Representative"
      .CellCaption(h0,2) = "Seattle"
      h0 = .AddItem("Andrew Fuller")
      .CellCaption(h0,1) = "Vice President, Sales"
      .CellCaption(h0,2) = "Tacoma"
```

```
      .SelectItem(h0) = 1
      h0 = .AddItem("Janet Leverling")
      .CellCaption(h0,1) = "Sales Representative"
      .CellCaption(h0,2) = "Kirkland"
      h0 = .AddItem("Margaret Peacock")
      .CellCaption(h0,1) = "Sales Representative"
      .CellCaption(h0,2) = "Redmond"
      h0 = .AddItem("Steven Buchanan")
      .CellCaption(h0,1) = "Sales Manager"
      .CellCaption(h0,2) = "London"
      h0 = .AddItem("Michael Suyama")
      .CellCaption(h0,1) = "Sales Representative"
      .CellCaption(h0,2) = "London"
      h0 = .AddItem("Robert King")
      .CellCaption(h0,1) = "Sales Representative"
      .CellCaption(h0,2) = "London"
      h0 = .AddItem("Laura Callahan")
      .CellCaption(h0,1) = "Inside Sales Coordinator"
      .CellCaption(h0,2) = "Seattle"
      h0 = .AddItem("Anne Dodsworth")
      .CellCaption(h0,1) = "Sales Representative"
      .CellCaption(h0,2) = "London"
   End With
   .EndUpdate
End With
```

C++

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

   #import <ExTree.dll>
   using namespace EXTREELib;
*/
EXTREELib::ITreePtr spTree1 = GetDlgItem(IDC_TREE1)->GetControlUnknown();
spTree1->BeginUpdate();
```

```
spTree1->PutColumnAutoResize(VARIANT_TRUE);
spTree1->PutContinueColumnScroll(VARIANT_FALSE);
spTree1->PutMarkSearchColumn(VARIANT_FALSE);
spTree1->PutSearchColumnIndex(1);
spTree1->PutFilterBarPromptVisible(VARIANT_TRUE);
spTree1->PutFilterBarPromptPattern(L"london");
EXTREELib::IColumnsPtr var_Columns = spTree1->GetColumns();
   ((EXTREELib::IColumnPtr)(var_Columns->Add(L"Name")))->PutWidth(96);
   ((EXTREELib::IColumnPtr)(var_Columns->Add(L"Title")))->PutWidth(96);
   var_Columns->Add(L"City");
EXTREELib::IItemsPtr var_Items = spTree1->GetItems();
   long h0 = var_Items->AddItem("Nancy Davolio");
   var_Items->PutCellCaption(h0,long(1),"Sales Representative");
   var_Items->PutCellCaption(h0,long(2),"Seattle");
   h0 = var_Items->AddItem("Andrew Fuller");
   var_Items->PutCellCaption(h0,long(1),"Vice President, Sales");
   var_Items->PutCellCaption(h0,long(2),"Tacoma");
   var_Items->PutSelectItem(h0,VARIANT_TRUE);
   h0 = var_Items->AddItem("Janet Leverling");
   var_Items->PutCellCaption(h0,long(1),"Sales Representative");
   var_Items->PutCellCaption(h0,long(2),"Kirkland");
   h0 = var_Items->AddItem("Margaret Peacock");
   var_Items->PutCellCaption(h0,long(1),"Sales Representative");
   var_Items->PutCellCaption(h0,long(2),"Redmond");
   h0 = var_Items->AddItem("Steven Buchanan");
   var_Items->PutCellCaption(h0,long(1),"Sales Manager");
   var_Items->PutCellCaption(h0,long(2),"London");
   h0 = var_Items->AddItem("Michael Suyama");
   var_Items->PutCellCaption(h0,long(1),"Sales Representative");
   var_Items->PutCellCaption(h0,long(2),"London");
   h0 = var_Items->AddItem("Robert King");
   var_Items->PutCellCaption(h0,long(1),"Sales Representative");
   var_Items->PutCellCaption(h0,long(2),"London");
   h0 = var_Items->AddItem("Laura Callahan");
   var_Items->PutCellCaption(h0,long(1),"Inside Sales Coordinator");
   var_Items->PutCellCaption(h0,long(2),"Seattle");
   h0 = var_Items->AddItem("Anne Dodsworth");
```

```
    var_Items->PutCellCaption(h0,long(1),"Sales Representative");
    var_Items->PutCellCaption(h0,long(2),"London");
spTree1->EndUpdate();
```

C#

```
extree1.BeginUpdate();
extree1.ColumnAutoResize = true;
extree1.ContinueColumnScroll = false;
extree1.MarkSearchColumn = false;
extree1.SearchColumnIndex = 1;
extree1.FilterBarPromptVisible = true;
extree1.FilterBarPromptPattern = "london";
exontrol.EXTREELib.Columns var_Columns = extree1.Columns;
    (var_Columns.Add("Name") as exontrol.EXTREELib.Column).Width = 96;
    (var_Columns.Add("Title") as exontrol.EXTREELib.Column).Width = 96;
    var_Columns.Add("City");
exontrol.EXTREELib.Items var_Items = extree1.Items;
    int h0 = var_Items.AddItem("Nancy Davolio");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"Seattle");
    h0 = var_Items.AddItem("Andrew Fuller");
    var_Items.set_CellCaption(h0,1,"Vice President, Sales");
    var_Items.set_CellCaption(h0,2,"Tacoma");
    var_Items.set_SelectItem(h0,1);
    h0 = var_Items.AddItem("Janet Leverling");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"Kirkland");
    h0 = var_Items.AddItem("Margaret Peacock");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"Redmond");
    h0 = var_Items.AddItem("Steven Buchanan");
    var_Items.set_CellCaption(h0,1,"Sales Manager");
    var_Items.set_CellCaption(h0,2,"London");
    h0 = var_Items.AddItem("Michael Suyama");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"London");
```

```
    h0 = var_Items.AddItem("Robert King");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"London");
    h0 = var_Items.AddItem("Laura Callahan");
    var_Items.set_CellCaption(h0,1,"Inside Sales Coordinator");
    var_Items.set_CellCaption(h0,2,"Seattle");
    h0 = var_Items.AddItem("Anne Dodsworth");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"London");
extree1.EndUpdate();
```

C# for /COM

```
axTree1.BeginUpdate();
axTree1.ColumnAutoResize = true;
axTree1.ContinueColumnScroll = false;
axTree1.MarkSearchColumn = false;
axTree1.SearchColumnIndex = 1;
axTree1.FilterBarPromptVisible = true;
axTree1.FilterBarPromptPattern = "london";
EXTREELib.Columns var_Columns = axTree1.Columns;
    (var_Columns.Add("Name") as EXTREELib.Column).Width = 96;
    (var_Columns.Add("Title") as EXTREELib.Column).Width = 96;
    var_Columns.Add("City");
EXTREELib.Items var_Items = axTree1.Items;
    int h0 = var_Items.AddItem("Nancy Davolio");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"Seattle");
    h0 = var_Items.AddItem("Andrew Fuller");
    var_Items.set_CellCaption(h0,1,"Vice President, Sales");
    var_Items.set_CellCaption(h0,2,"Tacoma");
    var_Items.set_SelectItem(h0,true);
    h0 = var_Items.AddItem("Janet Leverling");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"Kirkland");
    h0 = var_Items.AddItem("Margaret Peacock");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
```

```
    var_Items.set_CellCaption(h0,2,"Redmond");
    h0 = var_Items.AddItem("Steven Buchanan");
    var_Items.set_CellCaption(h0,1,"Sales Manager");
    var_Items.set_CellCaption(h0,2,"London");
    h0 = var_Items.AddItem("Michael Suyama");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"London");
    h0 = var_Items.AddItem("Robert King");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"London");
    h0 = var_Items.AddItem("Laura Callahan");
    var_Items.set_CellCaption(h0,1,"Inside Sales Coordinator");
    var_Items.set_CellCaption(h0,2,"Seattle");
    h0 = var_Items.AddItem("Anne Dodsworth");
    var_Items.set_CellCaption(h0,1,"Sales Representative");
    var_Items.set_CellCaption(h0,2,"London");
axTree1.EndUpdate();
```

Delphi

```
with AxTree1 do
begin
    BeginUpdate();
    ColumnAutoResize := True;
    ContinueColumnScroll := False;
    MarkSearchColumn := False;
    SearchColumnIndex := 1;
    FilterBarPromptVisible := True;
    FilterBarPromptPattern := 'london';
    with Columns do
    begin
        (Add('Name') as EXTREELib.Column).Width := 96;
        (Add('Title') as EXTREELib.Column).Width := 96;
        Add('City');
    end;
    with Items do
    begin
```

```
    h0 := AddItem('Nancy Davolio');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Representative';
    CellCaption[TObject(h0),TObject(2)] := 'Seattle';
    h0 := AddItem('Andrew Fuller');
    CellCaption[TObject(h0),TObject(1)] := 'Vice President, Sales';
    CellCaption[TObject(h0),TObject(2)] := 'Tacoma';
    SelectItem[h0] := True;
    h0 := AddItem('Janet Leverling');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Representative';
    CellCaption[TObject(h0),TObject(2)] := 'Kirkland';
    h0 := AddItem('Margaret Peacock');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Representative';
    CellCaption[TObject(h0),TObject(2)] := 'Redmond';
    h0 := AddItem('Steven Buchanan');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Manager';
    CellCaption[TObject(h0),TObject(2)] := 'London';
    h0 := AddItem('Michael Suyama');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Representative';
    CellCaption[TObject(h0),TObject(2)] := 'London';
    h0 := AddItem('Robert King');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Representative';
    CellCaption[TObject(h0),TObject(2)] := 'London';
    h0 := AddItem('Laura Callahan');
    CellCaption[TObject(h0),TObject(1)] := 'Inside Sales Coordinator';
    CellCaption[TObject(h0),TObject(2)] := 'Seattle';
    h0 := AddItem('Anne Dodsworth');
    CellCaption[TObject(h0),TObject(1)] := 'Sales Representative';
    CellCaption[TObject(h0),TObject(2)] := 'London';
  end;
  EndUpdate();
end
```

VB

```
With Tree1
  .BeginUpdate
  .ColumnAutoResize = True
```

```
.ContinueColumnScroll = 0
.MarkSearchColumn = False
.SearchColumnIndex = 1
.FilterBarPromptVisible = True
.FilterBarPromptPattern = "london"
With .Columns
   .Add("Name").Width = 96
   .Add("Title").Width = 96
   .Add "City"
End With
With .Items
   h0 = .AddItem("Nancy Davolio")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "Seattle"
   h0 = .AddItem("Andrew Fuller")
   .CellCaption(h0,1) = "Vice President, Sales"
   .CellCaption(h0,2) = "Tacoma"
   .SelectItem(h0) = 1
   h0 = .AddItem("Janet Leverling")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "Kirkland"
   h0 = .AddItem("Margaret Peacock")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "Redmond"
   h0 = .AddItem("Steven Buchanan")
   .CellCaption(h0,1) = "Sales Manager"
   .CellCaption(h0,2) = "London"
   h0 = .AddItem("Michael Suyama")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "London"
   h0 = .AddItem("Robert King")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "London"
   h0 = .AddItem("Laura Callahan")
   .CellCaption(h0,1) = "Inside Sales Coordinator"
   .CellCaption(h0,2) = "Seattle"
   h0 = .AddItem("Anne Dodsworth")
```

```
    .CellCaption(h0,1) = "Sales Representative"
    .CellCaption(h0,2) = "London"
  End With
  .EndUpdate
End With
```

VB.NET

```
Dim h0
With Extree1
  .BeginUpdate()
  .ColumnAutoResize = True
  .ContinueColumnScroll = False
  .MarkSearchColumn = False
  .SearchColumnIndex = 1
  .FilterBarPromptVisible = True
  .FilterBarPromptPattern = "london"
  With .Columns
    .Add("Name").Width = 96
    .Add("Title").Width = 96
    .Add("City")
  End With
  With .Items
    h0 = .AddItem("Nancy Davolio")
    .set_CellCaption(h0,1,"Sales Representative")
    .set_CellCaption(h0,2,"Seattle")
    h0 = .AddItem("Andrew Fuller")
    .set_CellCaption(h0,1,"Vice President, Sales")
    .set_CellCaption(h0,2,"Tacoma")
    .set_SelectItem(h0,1)
    h0 = .AddItem("Janet Leverling")
    .set_CellCaption(h0,1,"Sales Representative")
    .set_CellCaption(h0,2,"Kirkland")
    h0 = .AddItem("Margaret Peacock")
    .set_CellCaption(h0,1,"Sales Representative")
    .set_CellCaption(h0,2,"Redmond")
    h0 = .AddItem("Steven Buchanan")
```

```
      .set_CellCaption(h0,1,"Sales Manager")
      .set_CellCaption(h0,2,"London")
      h0 = .AddItem("Michael Suyama")
      .set_CellCaption(h0,1,"Sales Representative")
      .set_CellCaption(h0,2,"London")
      h0 = .AddItem("Robert King")
      .set_CellCaption(h0,1,"Sales Representative")
      .set_CellCaption(h0,2,"London")
      h0 = .AddItem("Laura Callahan")
      .set_CellCaption(h0,1,"Inside Sales Coordinator")
      .set_CellCaption(h0,2,"Seattle")
      h0 = .AddItem("Anne Dodsworth")
      .set_CellCaption(h0,1,"Sales Representative")
      .set_CellCaption(h0,2,"London")
   End With
   .EndUpdate()
End With
```

VB.NET for /COM

```
Dim h0
With AxTree1
   .BeginUpdate()
   .ColumnAutoResize = True
   .ContinueColumnScroll = False
   .MarkSearchColumn = False
   .SearchColumnIndex = 1
   .FilterBarPromptVisible = True
   .FilterBarPromptPattern = "london"
   With .Columns
      .Add("Name").Width = 96
      .Add("Title").Width = 96
      .Add("City")
   End With
   With .Items
      h0 = .AddItem("Nancy Davolio")
      .CellCaption(h0,1) = "Sales Representative"
```

```
        .CellCaption(h0,2) = "Seattle"
        h0 = .AddItem("Andrew Fuller")
        .CellCaption(h0,1) = "Vice President, Sales"
        .CellCaption(h0,2) = "Tacoma"
        .SelectItem(h0) = True
        h0 = .AddItem("Janet Leverling")
        .CellCaption(h0,1) = "Sales Representative"
        .CellCaption(h0,2) = "Kirkland"
        h0 = .AddItem("Margaret Peacock")
        .CellCaption(h0,1) = "Sales Representative"
        .CellCaption(h0,2) = "Redmond"
        h0 = .AddItem("Steven Buchanan")
        .CellCaption(h0,1) = "Sales Manager"
        .CellCaption(h0,2) = "London"
        h0 = .AddItem("Michael Suyama")
        .CellCaption(h0,1) = "Sales Representative"
        .CellCaption(h0,2) = "London"
        h0 = .AddItem("Robert King")
        .CellCaption(h0,1) = "Sales Representative"
        .CellCaption(h0,2) = "London"
        h0 = .AddItem("Laura Callahan")
        .CellCaption(h0,1) = "Inside Sales Coordinator"
        .CellCaption(h0,2) = "Seattle"
        h0 = .AddItem("Anne Dodsworth")
        .CellCaption(h0,1) = "Sales Representative"
        .CellCaption(h0,2) = "London"
    End With
    .EndUpdate()
End With
```

VFP

```
with thisform.Tree1
    .BeginUpdate
    .ColumnAutoResize = .T.
    .ContinueColumnScroll = 0
    .MarkSearchColumn = .F.
```

```
.SearchColumnIndex = 1
.FilterBarPromptVisible = .T.
.FilterBarPromptPattern = "london"
with .Columns
   .Add("Name").Width = 96
   .Add("Title").Width = 96
   .Add("City")
endwith
with .Items
   h0 = .AddItem("Nancy Davolio")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "Seattle"
   h0 = .AddItem("Andrew Fuller")
   .CellCaption(h0,1) = "Vice President, Sales"
   .CellCaption(h0,2) = "Tacoma"
   .SelectItem(h0) = 1
   h0 = .AddItem("Janet Leverling")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "Kirkland"
   h0 = .AddItem("Margaret Peacock")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "Redmond"
   h0 = .AddItem("Steven Buchanan")
   .CellCaption(h0,1) = "Sales Manager"
   .CellCaption(h0,2) = "London"
   h0 = .AddItem("Michael Suyama")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "London"
   h0 = .AddItem("Robert King")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "London"
   h0 = .AddItem("Laura Callahan")
   .CellCaption(h0,1) = "Inside Sales Coordinator"
   .CellCaption(h0,2) = "Seattle"
   h0 = .AddItem("Anne Dodsworth")
   .CellCaption(h0,1) = "Sales Representative"
   .CellCaption(h0,2) = "London"
```

```
    endwith
    .EndUpdate
endwith
```

property Tree.FilterBarPromptType as FilterPromptEnum

Specifies the type of the filter prompt.

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

By default, the FilterBarPromptType property is exFilterPromptContainsAll. The filter prompt feature allows you to filter the items as you type while the filter bar is visible on the bottom part of the list area. The Filter prompt feature allows at runtime filtering data on hidden columns too. Use the FilterBarPromptVisible property to show the filter prompt. Use the FilterBarPrompt property to specify the HTML caption being displayed in the filter bar when the filter pattern is missing. The FilterBarPromptPattern property specifies the pattern to filter the list. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered. Use the FilterBarCaption property to change the caption in the filter bar once a new filter is applied.

The FilterBarPromptType property supports the following values:

- **exFilterPromptContainsAll**, The list includes the items that contains all specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptContainsAny**, The list includes the items that contains any of specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptStartWith**, The list includes the items that starts with any specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptEndWith**, The list includes the items that ends with any specified sequences in the filter (FilterBarPromptPattern property). Can be combined with exFilterPromptCaseSensitive, exFilterPromptStartWords, exFilterPromptEndWords or exFilterPromptWords
- **exFilterPromptPattern**, The filter indicates a pattern that may include wild characters to be used to filter the items in the list. The FilterBarPromptPattern property may

include wild characters as follows:

- '?' for any single character
- '*' for zero or more occurrences of any character
- '#' for any digit character
- ' ' space delimits the patterns inside the filter

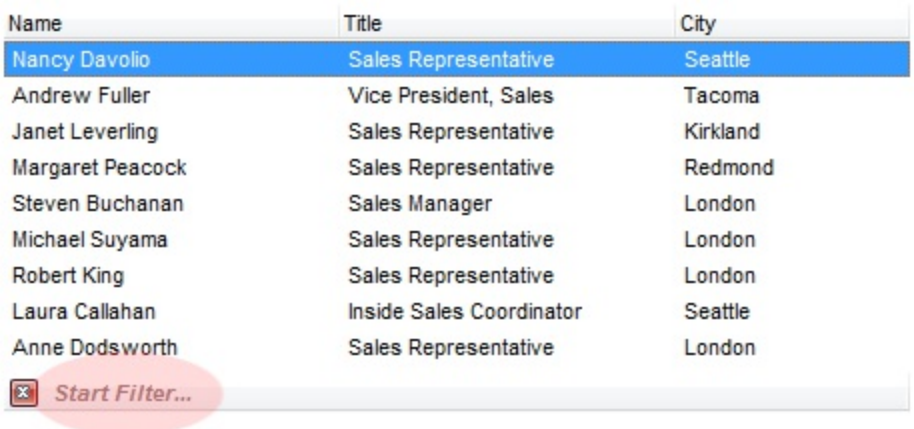
property Tree.FilterBarPromptVisible as FilterBarVisibleEnum

Shows or hides the control's filter bar including filter prompt.

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

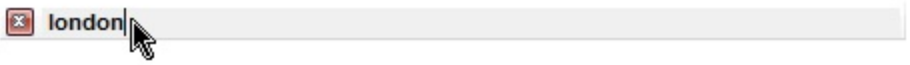
By default, The FilterBarPromptVisible property is exFilterBarHidden. The filter prompt feature allows you to filter the items as you type while the filter bar is visible on the bottom part of the list area. The Filter prompt feature allows at runtime filtering data on hidden columns too. Use the FilterBarPromptVisible property to show the filter prompt. Use the FilterBarPrompt property to specify the HTML caption being displayed in the filter bar when the filter pattern is missing. The FilterBarPromptPattern property specifies the pattern to filter the list. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way. The FilterBarCaption property defines the caption to be displayed on the control's filter bar. The FilterBarPromptType property specifies the type of filtering when the user edits the prompt in the filter bar. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered.

The following screen show shows the filter prompt:



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

Name	Title	City
Steven Buchanan	Sales Manager	London
Michael Suyama	Sales Representative	London
Robert King	Sales Representative	London
Anne Dodsworth	Sales Representative	London



property Tree.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 treeed 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 treeed 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.

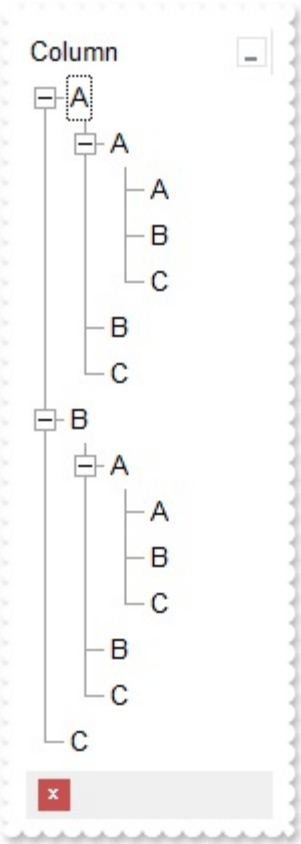
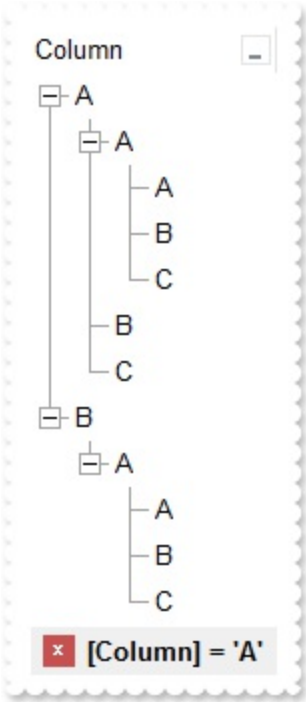
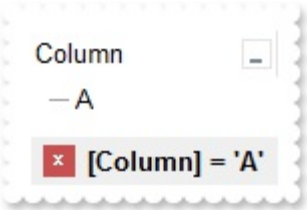
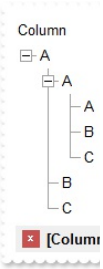
property Tree.FilterInclude as FilterIncludeEnum

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 exItemsWithoutChilds, 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	exItemsWithoutChilds 0	exItemsWithChilds 1	exRootsWithoutChilds 2	exRootsW 3
				

property Tree.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object used to paint the items.

Use the Font property to change the control's font . Use the FilterBarFont property to assign a different font for the control's filter bar. Use the Refresh method to refresh the control. Use the BeginUpdate and EndUpdate method to maintain performance while adding new columns or items.

The following VB sample assigns by code a new font to the control:

```
With Tree1
   With .Font
      .Name = "Tahoma"
   End With
   .Refresh
End With
```

The following C++ sample assigns by code a new font to the control:

```
COleFont font = m_tree.GetFont();
font.SetName( "Tahoma" );
m_tree.Refresh();
```

the C++ sample requires definition of COleFont class (#include "Font.h")

The following VB.NET sample assigns by code a new font to the control:

```
With AxTree1
   Dim font As System.Drawing.Font = New System.Drawing.Font("Tahoma", 10,
FontStyle.Regular, GraphicsUnit.Point)
   .Font = font
   .CtlRefresh()
End With
```

The following C# sample assigns by code a new font to the control:

```
System.Drawing.Font font = new System.Drawing.Font("Tahoma", 10, FontStyle.Regular);
```

```
axTree1.Font = font;
axTree1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the control:

```
with thisform.Tree1.Object
   .Font.Name = "Tahoma"
   .Refresh()
endwith
```

The following Template sample assigns by code a new font to the control:

```
Font
{
   Name = "Tahoma"
}
```

property Tree.ForeColor as Color

Retrieves or sets a value that indicates the control's foreground color.

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

The ForeColor property changes the foreground color of the control's scrolled area. The ExTree control can group the columns into two categories: locked and unlocked. The Locked category contains all the columns that are fixed to the left area of the client area. These columns cannot be scrolled horizontally. Use the CountLockedColumns to specify the number of locked columns. The unlocked are contains the columns that can be scrolled horizontally. To change the background color of the control's locked area use BackColorLock property. Use the CellForeColor property to specify the cell's foreground color. Use the ItemForeColor property to specify the item's foreground color.

property Tree.ForeColorHeader as Color

Specifies the header's foreground color.

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

Use the BackColorHeader and ForeColorHeader properties to customize the control's header. Use the Font property to change the control's font. Use the Add method to add new columns to the control. 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 HeaderVisible property to hide the control's header bar.

property Tree.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 control's foreground color for the locked area.

The ExTree 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.

property Tree.ForeColorSortBar as Color

Retrieves or sets a value that indicates the sort bar's foreground color.

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

Use the ForeColorSortBar property to specify the foreground color of the caption in the control's sort bar. Use the SortBarVisible property to show the control's sort bar. Use the SortBarCaption property to specify the caption of the sort bar, when the control's sort bar contains no columns. Use the BackColorSortBar property to specify the background color of the control's sort bar. Use the BackColorSortBarCaption property to specify the caption's background color in the control's sort bar. Use the ForeColor property to specify the control's foreground color. Use the ForeColorHeader property to specify the background color of the control's header bar.

method Tree.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 Tree.

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 Tree.FormatAnchor(New as Boolean) as String

Specifies the visual effect for anchor elements in HTML captions.

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

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

method Tree.FreezeEvents (Freeze as Boolean)

Prevents the control to fire any event.

Type	Description
Freeze as Boolean	A Boolean expression that specifies whether the control' events are froze or unfroze

The FreezeEvents(True) method freezes the control's events until the FreezeEvents(False) method is called. You can use the FreezeEvents method to improve performance of the control while loading data into it. For instance, the AddItem event occurs once a new item is added to the control's list, so during init time, you can call FreezeEvents(True) before, and FreezeEvents(False) after initialization is done.

The following samples show how you can lock the events while adding columns, items to the control:

```
' AddItem event occurs once a new item is added to the control's list
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   With Tree1
      Debug.Print ("AddItem event")
   End With
End Sub


With Tree1
   .FreezeEvents True
   .BeginUpdate
   With .Columns
      .Add("C1").Def(exCellHasCheckBox) = True
      .Add "C2"
   End With
   With .Items
      .CellCaption(.AddItem("SubItem 1.1"), 1) = "SubItem 1.2"
      .CellCaption(.AddItem("SubItem 2.1"), 1) = "SubItem 2.2"
   End With
   .EndUpdate
   .FreezeEvents False
End With
```

property Tree.FullRowSelect as Boolean

Enables full-row selection in the control.

Type	Description
Boolean	A boolean expression that indicates whether the control support full-row selection.

The FullRowSelect property specifies whether the selection spans the entire width of the control. The column pointed by the SelectColumnIndex specifies the column where the selected cell is marked. Use the SelectItem property to select programmatically an item. Use the SingleSel property to allow multiple items selection.

method Tree.GetItems (Options as Variant)

Gets the collection of items into a safe array,

Type	Description
Options as Variant	Specifies a long expression as follows: • if **0**, the result is a two-dimensional array with cell's captions. The list includes the *collapsed* 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 *does not include the collapsed items*. 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 *collapsed* 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. If the Options parameter is 0, the GetItems method collect the child items as well, no matter if the parent item is collapsed or expanded. 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[])extree1.GetItems()
```

or when the control contains multiple columns, the syntax is as follows:

```
object[,] Items = (object[,])extree1.GetItems()
```

The following VB.NET sample converts the returned value to a Object() when the control contains a single column:

```
Dim Items As Object() = Extree1.GetItems()
```

or when the control contains multiple columns, the syntax is as follows:

```
Dim Items As Object(,) = Extree1.GetItems()
```

/COM version:

The following VB sample gets the items from a control and put them to the second one:

```
With Tree2
  .BeginUpdate
    .Columns.Clear
    Dim c As EXTREELibCtl.Column
    For Each c In Tree1.Columns
      .Columns.Add c.Caption
    Next
    .PutItems Tree1.GetItems
```

```
    .EndUpdate
End With
```

The following C++ sample gets the items from a control an put to the second one:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_tree2.BeginUpdate();
    CColumns columns = m_tree.GetColumns(), columns2 = m_tree2.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_tree.GetItems( vtMissing );
    m_tree2.PutItems( &vtItems, vtMissing );
m_tree2.EndUpdate();
```

The following C# sample gets the items from a control and put them to a second one:

```
axTree2.BeginUpdate();
for (int i = 0; i < axTree1.Columns.Count; i++)
    axTree2.Columns.Add(axTree1.Columns[i].Caption);
object vtItems = axTree1.GetItems("");
axTree2.PutItems(ref vtItems);
axTree2.EndUpdate();
```

The following VB.NET sample gets the items from a control and put them to a second one:

```
With AxTree2
    .BeginUpdate()
    Dim j As Integer
    For j = 0 To AxTree1.Columns.Count - 1
        .Columns.Add(AxTree1.Columns(j).Caption)
    Next
    Dim vtItems As Object
    vtItems = AxTree1.GetItems("")
    .PutItems(vtItems)
    .EndUpdate()
End With
```

The following VFP sample gets the items from a control and put them to a second one:

```
local i
with thisform.Tree2
   .BeginUpdate()
   for i = 0 to thisform.Tree1.Columns.Count - 1
      .Columns.Add( thisform.Tree1.Columns(i).Caption )
   next
   local array vtItems[1]
   vtItems = thisform.Tree1.GetItems("")
   .PutItems( @vtItems )
   .EndUpdate()
endwith
```

property Tree.GridLineColor as Color

Specifies the grid line color.

Type	Description
Color	A color expression that indicates the color of the grid lines.

Use the GridLineColor property to specify the color for grid lines. Use the DrawGridLines property to show the grid lines. Use the 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 Tree.GridLineStyle as GridLinesStyleEnum

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

Type	Description
GridLinesStyleEnum	A GridLinesStyleEnum expression that specifies the style to show the control's horizontal or vertical lines.

By default, the GridLineStyle property is exGridLinesDot. The GridLineStyle property has effect only if the DrawGridLines property is not zero. The GridLineStyle property can be used to specify the style for horizontal or/and vertical grid lines. Use the GridLineColor property to specify the color for 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 = GridLinesStyleEnum.exGridLinesHDash Or
GridLinesStyleEnum.exGridLinesVSolid
```

The following VB/NET sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXTREELib.GridLinesStyleEnum.exGridLinesHDash Or
exontrol.EXTREELib.GridLinesStyleEnum.exGridLinesVSolid
```

The following C# sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXTREELib.GridLinesStyleEnum.exGridLinesHDash |
exontrol.EXTREELib.GridLinesStyleEnum.exGridLinesVSolid;
```

The following Delphi sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle := Integer(EXTREELib.GridLinesStyleEnum.exGridLinesHDash) Or
Integer(EXTREELib.GridLinesStyleEnum.exGridLinesVSolid);
```

The following VFP sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = 36
```

The following screen shot shows the control using different grid lines for columns and chart area:

property Tree.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 as an alternative to double-clicking the parent item, in case the ExpandOnDblClick property is True. 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 ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample changes the +/- button appearance:

```
With Tree1
   .HasButtons = ExpandButtonEnum.exWPlus
End With
```

The following C++ sample changes the +/- button appearance:

```
m_tree.SetHasButtons( 3 /*exWPlus*/ );
```

The following VB.NET sample changes the +/- button appearance:

```
With AxTree1
   .HasButtons = EXTREELib.ExpandButtonEnum.exWPlus
End With
```

The following C# sample changes the +/- button appearance:

```
axTree1.HasButtons = EXTREELib.ExpandButtonEnum.exWPlus;
```

The following VFP sample changes the +/- button appearance:

```
with thisform.Tree1
```

```
    .HasButtons = 3 && exWPlus
endwith
```

property Tree.HasButtonsCustom(Expanded as Boolean) as Long

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

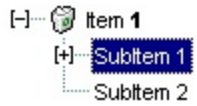
Type	Description
Expanded as Boolean	A boolean expression that indicates the sign being changed.
Long	A long expression that indicates the icon being used for +/- signs on the parent items. The last 7 bits in the high significant byte of the long expression indicates the identifier of the skin being 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, ReplaceIcon methods to add new icons to the control, at runtime. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample specifies different (as in the screen shot) +/- signs for the control:

```
With Tree1
   .BeginUpdate
      .Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExm

      .LinesAtRoot = exLinesAtRoot
      .HeaderVisible = False
      .HasButtons = exCustom
      .HasButtonsCustom(False) = 1
      .HasButtonsCustom(True) = 2
      .Columns.Add "Column 1"
      With .Items
         Dim h As HITEM
```

```
        h = .AddItem("Item 1")
        .InsertItem h, , "SubItem 1"
        .InsertItem h, , "SubItem 2"
      End With
   .EndUpdate
End With
```



The following C++ sample specifies different (as in the screen shot) +/- signs for the control:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_tree.BeginUpdate();
m_tree.Images( COleVariant(
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExm
 ) );
m_tree.SetLinesAtRoot( -1 );
m_tree.SetHeaderVisible( FALSE );
m_tree.SetHasButtons( 4 /*exCustom*/ );
m_tree.SetHasButtonsCustom( FALSE, 1 );
m_tree.SetHasButtonsCustom( TRUE, 2 );
m_tree.GetColumns().Add( "Column 1" );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItems items = m_tree.GetItems();
long h = items.AddItem( COleVariant( "Item 1" ) );
items.InsertItem( h, vtMissing, COleVariant( "SubItem 1" ) );
items.InsertItem( h, vtMissing, COleVariant( "SubItem 2" ) );
m_tree.EndUpdate();
```

The following VB.NET sample specifies different (as in the screen shot) +/- signs for the control:

```
With AxTree1
   .BeginUpdate()
```

```
.Images("gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlkt

    .LinesAtRoot = EXTREELib.LinesAtRootEnum.exLinesAtRoot
    .HeaderVisible = False
    .HasButtons = EXTREELib.ExpandButtonEnum.exCustom
    .set_HasButtonsCustom(False, 1)
    .set_HasButtonsCustom(True, 2)
    .Columns.Add("Column 1")
    With .Items
        Dim h As Long
        h = .AddItem("Item 1")
        .InsertItem(h, , "SubItem 1")
        .InsertItem(h, , "SubItem 2")
    End With
    .EndUpdate()
End With
```

The following C# sample specifies different (as in the screen shot) +/- signs for the control:

```
axTree1.BeginUpdate();
axTree1.Images("gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0

axTree1.LinesAtRoot = EXTREELib.LinesAtRootEnum.exLinesAtRoot;
axTree1.HeaderVisible = false;
axTree1.HasButtons = EXTREELib.ExpandButtonEnum.exCustom;
axTree1.set_HasButtonsCustom(false, 1);
axTree1.set_HasButtonsCustom(true, 2);
axTree1.Columns.Add("Column 1");
int h = axTree1.Items.AddItem("Item 1");
axTree1.Items.InsertItem(h, "", "SubItem 1");
axTree1.Items.InsertItem(h, "", "SubItem 2");
axTree1.EndUpdate();
```

The following VFP sample specifies different (as in the screen shot) +/- signs for the control:

```
with thisform.Tree1
```

```
   .BeginUpdate()
      local s
      s =
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExm

      s = s +
"1ntFptVrtItt1vuFxuVzul1u13vF5vV7vl9v1/wGBwWDwmFw2HxGJxWLxmNx2PyGRyWTymV

      .Images(s)
      .LinesAtRoot = -1
      .HeaderVisible = .f.
      .HasButtons = 4 &&exCustom
      local sT, sCR
      sCR = chr(13) + chr(10)
      sT = "HasButtonsCustom(True) = 2"+ sCR
      sT = sT + "HasButtonsCustom(False) = 1"+ sCR
      .Template = sT
      .Columns.Add("Column 1")
      With .Items
         local h
         h = .AddItem("Item 1")
         .InsertItem(h, , "SubItem 1")
         .InsertItem(h, , "SubItem 2")
      EndWith
   .EndUpdate()
endwith
```

property Tree.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 uses the lines to link the items of the hierarchy.

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. Use the InsertControlItem property to insert an ActiveX item.

property Tree.HeaderAppearance as AppearanceEnum

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

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the header's appearance.

Use the HeaderAppearance property to change the appearance of the control's header bar. Use the HeaderVisible property to hide the control's header bar. Use the Appearance property to specify the control's appearance.

property Tree.HeaderEnabled as Boolean

Enables or disables the control's header.

Type	Description
Boolean	A boolean expression that specifies whether the control's header is enabled or disabled.

By default, the HeaderEnabled property is True. The HeaderEnabled property enables or disables the control's header (including the control's sort/groupby-bar). If the header is disabled, the user can't resize, sort or drag and drop any column. Also, if the header is disabled, the control's sort/groupby-bar is disabled as well. The HeaderVisible property shows or hides the control's header. The SortBarVisible property shows or hides the control's sort/groupby-bar.

property Tree.HeaderHeight as Long

Retrieves or sets a value indicating the control's header height.

Type	Description
Long	A long expression that indicates the height of the control's header bar.

By default, the HeaderHeight property is 18 pixels. Use the HeaderHeight property to change the height of the control's header bar. Use the HeaderVisible property to hide the control's header bar. Use the LevelKey property to display the control's header bar using multiple levels. If the control displays the header bar using multiple levels the HeaderHeight property gets the height in pixels of a single level in the header bar. The control's header displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the HTMLCaption property to display multiple lines in the column's caption. Use the Add method to add new columns to the control. Use the LevelKey property to specify columns on the same level. *If the HeaderSingleLine property is False, the HeaderHeight property specifies the maximum height of the control's header when the user resizes the columns.*

The following VB sample displays a header bar using multiple lines:

```
With Tree1
    .BeginUpdate
      .HeaderHeight = 32
      With .Columns.Add("Column 1")
        .HTMLCaption = "Line1<br>Line2"
      End With
      With .Columns.Add("Column 2")
        .HTMLCaption = "Line1<br>Line2"
      End With
    .EndUpdate
End With
```

The following C++ sample displays a header bar using multiple lines:

```
#include "Columns.h"
#include "Column.h"
m_tree.BeginUpdate();
m_tree.SetHeaderHeight( 32 );
m_tree.SetHeaderVisible( TRUE );
```

```
CColumn column1( V_DISPATCH( &m_tree.GetColumns().Add( "Column 1" ) ) );
   column1.SetHTMLCaption( "Line1<br>Line2" );
CColumn column2( V_DISPATCH( &m_tree.GetColumns().Add( "Column 2" ) ) );
   column2.SetHTMLCaption( "Line1<br>Line2" );
m_tree.EndUpdate();
```

The following VB.NET sample displays a header bar using multiple lines:

```
With AxTree1
   .BeginUpdate()
   .HeaderVisible = True
   .HeaderHeight = 32
   With .Columns.Add("Column 1")
      .HTMLCaption = "Line1<br>Line2"
   End With
   With .Columns.Add("Column 2")
      .HTMLCaption = "Line1<br>Line2"
   End With
   .EndUpdate()
End With
```

The following C# sample displays a header bar using multiple lines:

```
axTree1.BeginUpdate();
axTree1.HeaderVisible = true;
axTree1.HeaderHeight = 32;
EXTREELib.Column column1 = axTree1.Columns.Add("Column 1") as EXTREELib.Column ;
column1.HTMLCaption = "Line1<br>Line2";
EXTREELib.Column column2 = axTree1.Columns.Add("Column 2") as EXTREELib.Column;
column2.HTMLCaption = "Line1<br>Line2";
axTree1.EndUpdate();
```

The following VFP sample displays a header bar using multiple lines:

```
with thisform.Tree1
   .BeginUpdate()
   .HeaderVisible = .t.
   .HeaderHeight = 32
   with .Columns.Add("Column 1")
```

```
    .HTMLCaption = "Line1<br>Line2"
  endwith
  with .Columns.Add("Column 2")
    .HTMLCaption = "Line1<br>Line2"
  endwith
  .EndUpdate()
endwith
```

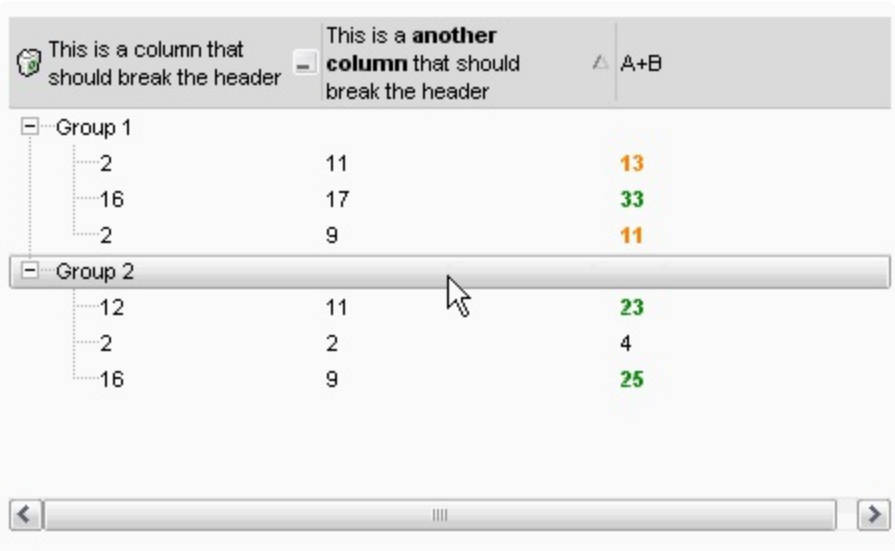
property Tree.HeaderSingleLine as Boolean

Specifies whether the control resizes the columns header and wraps the captions in single or multiple lines.

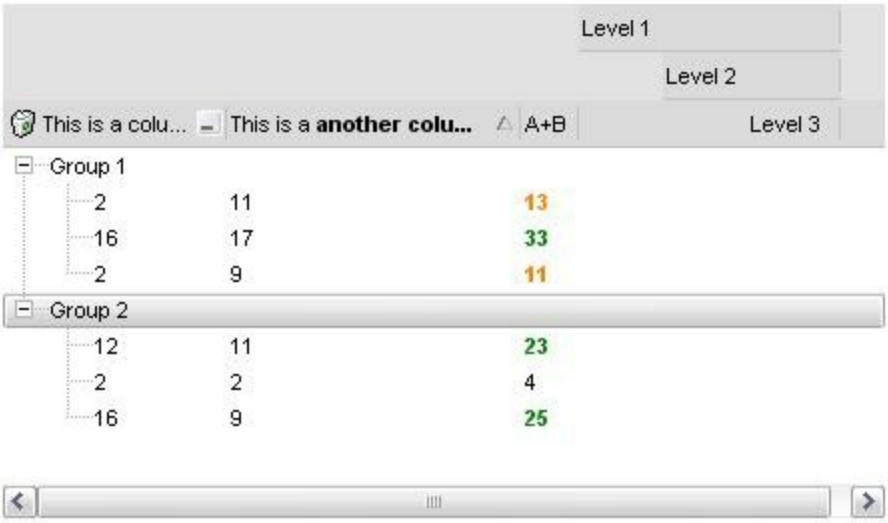
Type	Description
Boolean	A boolean expression that specifies whether the header displays single or multiple lines.

By defauly, the HeaderSingleLine property is True. If the HeaderSingleLine property is False the control breaks the column's caption as soon as the user resizes the column. **In this case the HeaderHeight property specifies the maximum height of the control's header.** The initial height is computed based on the control's Font property. The Caption property specifies the caption of the column being displayed in the control's header. The HTMLCaption property specifies the HTML caption of the column being displayed in the column's header. Use the LevelKey property to display the control's header on multiple levels.

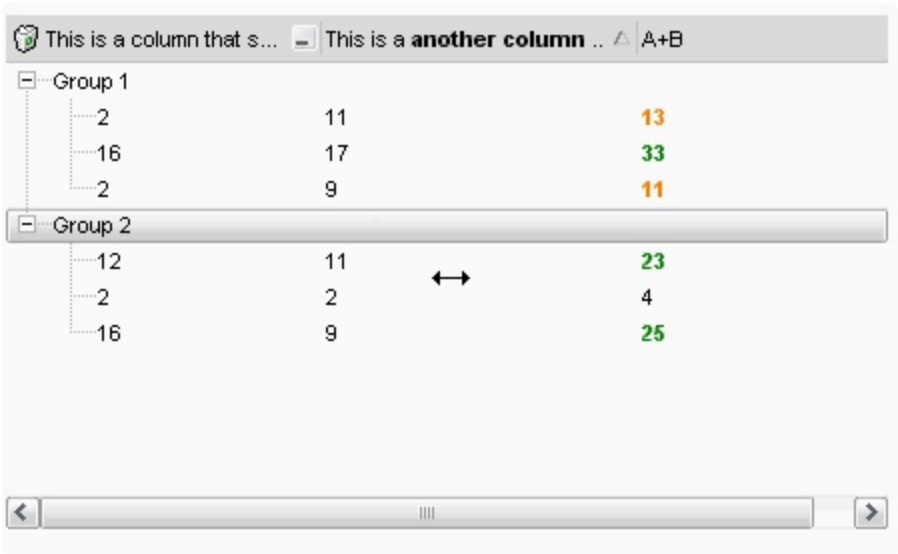
The following screen show shows the control's header while it displays a multiple lines (HeaderSingleLine = False):



The following screen shot shows the control's header on multiple levels using the LevelKey property:



The following screen show shows the control's header while it displays a single line (HeaderSingleLine = True):



property Tree.HeaderVisible as Boolean

Retrieves or sets a value that indicates whether the the tree's header is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the the tree's header is visible or hidden.

By default, the HeaderVisible property is True. Use the HeaderVisible property to hide the control's header bar. Use the HeaderAppearance property to change the header bar's appearance. Use the BackColorHeader and ForeColorHeader properties to customize the control's header. Use the BackColorLevelHeader property to specify the background color of the header when it displays multiple levels. Use the HeaderHeight property to specify the height of the control's header bar.

property Tree.HideSelection as Boolean

Returns a value that determines whether selected item appears highlighted when a control loses the focus.

Type	Description
Boolean	A boolean expression that indicates whether the selected item appears highlighted when a control loses the focus.

By default, the HideSelection property is False. You can use this property to indicate which item is highlighted while another form or a dialog box has the focus. Use the SelForeColor and SelBackColor property to customize the colors for the selected items in the control. Use the SelectItem property to programmatically select an item. Use the SelectedItem and SelectCount property to retrieve the list of selected items. Use the SelectableItem property to specify whether an items can be selected.

property Tree.HotBackColor as Color

Retrieves or sets a value that indicates the hot-tracking background color.

Type	Description
Color	A color expression that indicates the background color for item from the cursor (hovering the item). Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

By default, the HotBackColor property is 0, which means that the HotBackColor property has no effect. Use the HotBackColor property on a non-zero value to highlight the item from the cursor. The HotForeColor property specifies the foreground color to highlight the item from the cursor. The ItemFromPoint property gets the item from the cursor. The SelBackColor property specifies the selection background color. The SelBackMode property specifies the way the selected items are shown in the control.

The following sample displays a different background color mouse passes over an item.

VBA

```
With Tree1
   .BeginUpdate
   .Columns.Add "Def"
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .AddItem "Item A"
      .AddItem "Item B"
      .AddItem "Item C"
   End With
   .EndUpdate
End With
```

VB6

```
With Tree1
   .BeginUpdate
   .Columns.Add "Def"
```

```
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .AddItem "Item A"
      .AddItem "Item B"
      .AddItem "Item C"
   End With
   .EndUpdate
End With
```

VB.NET

```
With Exgantt1
   .BeginUpdate()
   .Columns.Add("Def")
   .HotBackColor = Color.FromArgb(0,0,128)
   .HotForeColor = Color.FromArgb(255,255,255)
   With .Items
      .AddItem("Item A")
      .AddItem("Item B")
      .AddItem("Item C")
   End With
   .EndUpdate()
End With
```

VB.NET for /COM

```
With AxTree1
   .BeginUpdate()
   .Columns.Add("Def")
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .AddItem("Item A")
      .AddItem("Item B")
      .AddItem("Item C")
   End With
   .EndUpdate()
```

```
End With
```

C++

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

   #import <ExTree.dll>
   using namespace EXGANTTLib;
*/
EXGANTTLib::ITreePtr spTree1 = GetDlgItem(IDC_GANTT1)->GetControlUnknown();
spTree1->BeginUpdate();
spTree1->GetColumns()->Add(L"Def");
spTree1->PutHotBackColor(RGB(0,0,128));
spTree1->PutHotForeColor(RGB(255,255,255));
EXGANTTLib::IItemsPtr var_Items = spTree1->GetItems();
   var_Items->AddItem("Item A");
   var_Items->AddItem("Item B");
   var_Items->AddItem("Item C");
spTree1->EndUpdate();
```

C++ Builder

```
Tree1->BeginUpdate();
Tree1->Columns->Add(L"Def");
Tree1->HotBackColor = RGB(0,0,128);
Tree1->HotForeColor = RGB(255,255,255);
Exganttlib_tlb::IItemsPtr var_Items = Tree1->Items;
   var_Items->AddItem(TVariant("Item A"));
   var_Items->AddItem(TVariant("Item B"));
   var_Items->AddItem(TVariant("Item C"));
Tree1->EndUpdate();
```

C#

```
exgantt1.BeginUpdate();
exgantt1.Columns.Add("Def");
```

```
exgantt1.HotBackColor = Color.FromArgb(0,0,128);
exgantt1.HotForeColor = Color.FromArgb(255,255,255);
exontrol.EXGANTTLib.Items var_Items = exgantt1.Items;
   var_Items.AddItem("Item A");
   var_Items.AddItem("Item B");
   var_Items.AddItem("Item C");
exgantt1.EndUpdate();
```

JavaScript

```
<OBJECT classid="clsid:CD481F4D-2D25-4759-803F-752C568F53B7" id="Tree1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   Tree1.BeginUpdate()

   Tree1.Columns.Add("Def")

   Tree1.HotBackColor = 8388608

   Tree1.HotForeColor = 16777215

   var var_Items = Tree1.Items

      var_Items.AddItem("Item A")

      var_Items.AddItem("Item B")

      var_Items.AddItem("Item C")

   Tree1.EndUpdate()

</SCRIPT>
```

C# for /COM

```
axTree1.BeginUpdate();
axTree1.Columns.Add("Def");
```

```
axTree1.HotBackColor = Color.FromArgb(0,0,128);
axTree1.HotForeColor = Color.FromArgb(255,255,255);
EXGANTTLib.Items var_Items = axTree1.Items;
   var_Items.AddItem("Item A");
   var_Items.AddItem("Item B");
   var_Items.AddItem("Item C");
axTree1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   COM com_Items

   anytype var_Items




   super()


   exgantt1.BeginUpdate()

   exgantt1.Columns().Add("Def")

   exgantt1.HotBackColor(WinApi::RGB2int(0,0,128))

   exgantt1.HotForeColor(WinApi::RGB2int(255,255,255))

   var_Items = exgantt1.Items()
    com_Items = var_Items

      com_Items.AddItem("Item A")

      com_Items.AddItem("Item B")
```

```
        com_Items.AddItem("Item C")

    exgantt1.EndUpdate()

}
```

VFP

```
with thisform.Tree1
    .BeginUpdate
    .Columns.Add("Def")
    .HotBackColor = RGB(0,0,128)
    .HotForeColor = RGB(255,255,255)
    with .Items
        .AddItem("Item A")
        .AddItem("Item B")
        .AddItem("Item C")
    endwith
    .EndUpdate
endwith
```

dBASE Plus

```
local oTree,var_Items

oTree = form.Activex1.nativeObject
oTree.BeginUpdate()
oTree.Columns.Add("Def")
oTree.HotBackColor = 0x800000
oTree.HotForeColor = 0xffffff
var_Items = oTree.Items
    var_Items.AddItem("Item A")
    var_Items.AddItem("Item B")
    var_Items.AddItem("Item C")
oTree.EndUpdate()
```

XBasic (Alpha Five)

```
Dim oTree as P
Dim var_Items as P

oTree = topparent:CONTROL_ACTIVEX1.activex
oTree.BeginUpdate()
oTree.Columns.Add("Def")
oTree.HotBackColor = 8388608
oTree.HotForeColor = 16777215
var_Items = oTree.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oTree.EndUpdate()
```

Delphi 8 (.NET only)

```
with AxTree1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := Color.FromArgb(0,0,128);
   HotForeColor := Color.FromArgb(255,255,255);
   with Items do
   begin
      AddItem('Item A');
      AddItem('Item B');
      AddItem('Item C');
   end;
   EndUpdate();
end
```

Delphi (standard)

```
with Tree1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := RGB(0,0,128);
```

```
   HotForeColor := RGB(255,255,255);
   with Items do
   begin
      AddItem('Item A');
      AddItem('Item B');
      AddItem('Item C');
   end;
   EndUpdate();
end
```

Visual Objects

```
local var_Items as IItems

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:Columns:Add("Def")
oDCOCX_Exontrol1:HotBackColor := RGB(0,0,128)
oDCOCX_Exontrol1:HotForeColor := RGB(255,255,255)
var_Items := oDCOCX_Exontrol1:Items
   var_Items:AddItem("Item A")
   var_Items:AddItem("Item B")
   var_Items:AddItem("Item C")
oDCOCX_Exontrol1:EndUpdate()
```

PowerBuilder

```
OleObject oTree,var_Items

oTree = ole_1.Object
oTree.BeginUpdate()
oTree.Columns.Add("Def")
oTree.HotBackColor = RGB(0,0,128)
oTree.HotForeColor = RGB(255,255,255)
var_Items = oTree.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oTree.EndUpdate()
```

property Tree.HotForeColor as Color

Retrieves or sets a value that indicates the hot-tracking foreground color.

Type	Description
Color	A color expression that indicates the foreground color for item from the cursor (hovering the item).

By default, the HotForeColor property is 0, which means that the HotForeColor property has no effect. Use the HotForeColor property on a non-zero value to highlight the item from the cursor. The HotBackColor property specifies the background color to highlight the item from the cursor. The ItemFromPoint property gets the item from the cursor. The SelForeColor property specifies the selection foreground color.

The following sample displays a different background color mouse passes over an item.

VBA

```
With Tree1
   .BeginUpdate
   .Columns.Add "Def"
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .AddItem "Item A"
      .AddItem "Item B"
      .AddItem "Item C"
   End With
   .EndUpdate
End With
```

VB6

```
With Tree1
   .BeginUpdate
   .Columns.Add "Def"
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .AddItem "Item A"
```

```
      .AddItem "Item B"
      .AddItem "Item C"
   End With
   .EndUpdate
End With
```

VB.NET

```
With Exgantt1
   .BeginUpdate()
   .Columns.Add("Def")
   .HotBackColor = Color.FromArgb(0,0,128)
   .HotForeColor = Color.FromArgb(255,255,255)
   With .Items
      .AddItem("Item A")
      .AddItem("Item B")
      .AddItem("Item C")
   End With
   .EndUpdate()
End With
```

VB.NET for /COM

```
With AxTree1
   .BeginUpdate()
   .Columns.Add("Def")
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   With .Items
      .AddItem("Item A")
      .AddItem("Item B")
      .AddItem("Item C")
   End With
   .EndUpdate()
End With
```

C++

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

   #import <ExTree.dll>
   using namespace EXGANTTLib;
*/
EXGANTTLib::ITreePtr spTree1 = GetDlgItem(IDC_GANTT1)->GetControlUnknown();
spTree1->BeginUpdate();
spTree1->GetColumns()->Add(L"Def");
spTree1->PutHotBackColor(RGB(0,0,128));
spTree1->PutHotForeColor(RGB(255,255,255));
EXGANTTLib::IItemsPtr var_Items = spTree1->GetItems();
   var_Items->AddItem("Item A");
   var_Items->AddItem("Item B");
   var_Items->AddItem("Item C");
spTree1->EndUpdate();
```

C++ Builder

```
Tree1->BeginUpdate();
Tree1->Columns->Add(L"Def");
Tree1->HotBackColor = RGB(0,0,128);
Tree1->HotForeColor = RGB(255,255,255);
Exganttlib_tlb::IItemsPtr var_Items = Tree1->Items;
   var_Items->AddItem(TVariant("Item A"));
   var_Items->AddItem(TVariant("Item B"));
   var_Items->AddItem(TVariant("Item C"));
Tree1->EndUpdate();
```

C#

```
exgantt1.BeginUpdate();
exgantt1.Columns.Add("Def");
exgantt1.HotBackColor = Color.FromArgb(0,0,128);
exgantt1.HotForeColor = Color.FromArgb(255,255,255);
exontrol.EXGANTTLib.Items var_Items = exgantt1.Items;
   var_Items.AddItem("Item A");
```

```
   var_Items.AddItem("Item B");
   var_Items.AddItem("Item C");
exgantt1.EndUpdate();
```

JavaScript

```
<OBJECT classid="clsid:CD481F4D-2D25-4759-803F-752C568F53B7" id="Tree1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   Tree1.BeginUpdate()

   Tree1.Columns.Add("Def")

   Tree1.HotBackColor = 8388608

   Tree1.HotForeColor = 16777215

   var var_Items = Tree1.Items

      var_Items.AddItem("Item A")

      var_Items.AddItem("Item B")

      var_Items.AddItem("Item C")

   Tree1.EndUpdate()

</SCRIPT>
```

C# for /COM

```
axTree1.BeginUpdate();
axTree1.Columns.Add("Def");
axTree1.HotBackColor = Color.FromArgb(0,0,128);
axTree1.HotForeColor = Color.FromArgb(255,255,255);
EXGANTTLib.Items var_Items = axTree1.Items;
   var_Items.AddItem("Item A");
```

```
    var_Items.AddItem("Item B");
    var_Items.AddItem("Item C");
axTree1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Items

    anytype var_Items




    super()


    exgantt1.BeginUpdate()

    exgantt1.Columns().Add("Def")

    exgantt1.HotBackColor(WinApi::RGB2int(0,0,128))

    exgantt1.HotForeColor(WinApi::RGB2int(255,255,255))

    var_Items = exgantt1.Items()
     com_Items = var_Items

        com_Items.AddItem("Item A")

        com_Items.AddItem("Item B")

        com_Items.AddItem("Item C")

    exgantt1.EndUpdate()
```

```
}
```

VFP

```
with thisform.Tree1
   .BeginUpdate
   .Columns.Add("Def")
   .HotBackColor = RGB(0,0,128)
   .HotForeColor = RGB(255,255,255)
   with .Items
      .AddItem("Item A")
      .AddItem("Item B")
      .AddItem("Item C")
   endwith
   .EndUpdate
endwith
```

dBASE Plus

```
local oTree,var_Items

oTree = form.Activex1.nativeObject
oTree.BeginUpdate()
oTree.Columns.Add("Def")
oTree.HotBackColor = 0x800000
oTree.HotForeColor = 0xffffff
var_Items = oTree.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oTree.EndUpdate()
```

XBasic (Alpha Five)

```
Dim oTree as P
Dim var_Items as P

oTree = topparent:CONTROL_ACTIVEX1.activex
```

```
oTree.BeginUpdate()
oTree.Columns.Add("Def")
oTree.HotBackColor = 8388608
oTree.HotForeColor = 16777215
var_Items = oTree.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oTree.EndUpdate()
```

Delphi 8 (.NET only)

```
with AxTree1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := Color.FromArgb(0,0,128);
   HotForeColor := Color.FromArgb(255,255,255);
   with Items do
   begin
      AddItem('Item A');
      AddItem('Item B');
      AddItem('Item C');
   end;
   EndUpdate();
end
```

Delphi (standard)

```
with Tree1 do
begin
   BeginUpdate();
   Columns.Add('Def');
   HotBackColor := RGB(0,0,128);
   HotForeColor := RGB(255,255,255);
   with Items do
   begin
      AddItem('Item A');
```

```
    AddItem('Item B');
    AddItem('Item C');
  end;
  EndUpdate();
end
```

Visual Objects

```
local var_Items as IItems

oDCOCX_Exontrol1:BeginUpdate()
oDCOCX_Exontrol1:Columns:Add("Def")
oDCOCX_Exontrol1:HotBackColor := RGB(0,0,128)
oDCOCX_Exontrol1:HotForeColor := RGB(255,255,255)
var_Items := oDCOCX_Exontrol1:Items
  var_Items:AddItem("Item A")
  var_Items:AddItem("Item B")
  var_Items:AddItem("Item C")
oDCOCX_Exontrol1:EndUpdate()
```

PowerBuilder

```
OleObject oTree,var_Items

oTree = ole_1.Object
oTree.BeginUpdate()
oTree.Columns.Add("Def")
oTree.HotBackColor = RGB(0,0,128)
oTree.HotForeColor = RGB(255,255,255)
var_Items = oTree.Items
  var_Items.AddItem("Item A")
  var_Items.AddItem("Item B")
  var_Items.AddItem("Item C")
oTree.EndUpdate()
```

property Tree.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

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

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

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

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

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

The screen shot was generated using the following x-script:

```
BeginUpdate

HTMLPicture("pic1") = "c:/temp/editors.gif"
HTMLPicture("pic2") = "c:/temp/editpaste.gif"

MarkSearchColumn = False
ShowFocusRect = False
LinesAtRoot = -1
HeaderHeight = 38
BackColorHeader = RGB(255,255,255)
HeaderAppearance = 5
BackColor = RGB(255,255,255)

ConditionalFormats
{
   Add("%2 > 15")
   {
      Bold = True
      ForeColor = RGB(0,128,0)
      ApplyTo = 2
   }
   Add("%2 > 10 and %2 < 18")
   {
      Bold = True
      ForeColor = RGB(255,128,0)
      ApplyTo = 2
   }

}
Columns
{
   Add("A")
   {
```

```
      Editor.EditType = 4
      HTMLCaption="A pic1"
   }
   Add("B")
   {
      Editor.EditType = 4
      HTMLCaption="B pic2"
   }
   Add("A+B")
   {
      ComputedField = "%0 + %1"
      HTMLCaption = "A pic1 + B pic2"
      HeaderBold = True
      HeaderAlignment = 2
      Alignment = 2
   }
}
Items
{
   Dim h, h1
   h = InsertItem(,,"Group 1")
   CellCaptionFormat(h,2) = 1
   h1 = InsertItem(h,,16)
   CellCaption(h1,1) = 17
   h1 = InsertItem(h,,2)
   CellCaption(h1,1) = 11
   h1 = InsertItem(h,,2)
   CellCaption(h1,1) = 9
   ExpandItem(h) = True
   h = InsertItem(,,"Group 2")
   CellCaptionFormat(h,2) = 1
   h1 = InsertItem(h,,16)
   CellCaption(h1,1) = 9
   h1 = InsertItem(h,,12)
   CellCaption(h1,1) = 11
   h1 = InsertItem(h,,2)
   CellCaption(h1,1) = 2
```

```
    ExpandItem(h) = True
    SelectItem(h) = True
}
EndUpdate
```

property Tree.hWnd as Long

Retrieves the control's window handle.

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

Use the hWnd property to get the control's main window handle. Use the ItemWindowHost property to get the handle of the container window that host an item's ActiveX Control. 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 Tree.HyperLinkColor as Color

Specifies the hyperlink color.

Type	Description
Color	A color expression that specifies the hyperlink color.

Use the HyperLinkColor property to specify the color used when the cursor is over the hyperlink cells. A hyperlink cell has the CellHyperLink property true. The control fires the HyperLinkClick property when user clicks a cell that has the CellHyperLink property on True.

method Tree.Images (Handle as Variant)

Sets a runtime the control's image tree.

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

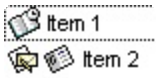
HIMAGELIST type. The GetSafeHandle() method of the CImageList gets the HIMAGELIST handle (long, loads icon from HIMAGELIST type)

The user can add images at design time, by drag and drop files to combo's image holder. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the RepaceIcon method to add, remove or clear icons in the control's images collection. Use the CellImage, CellImages properties to assign icons to a cell. Use the CellPicture property to assign a custom size picture to a cell. Use the CheckImage or RadioImage property to specify a different look for checkboxes or radio buttons in the cells.

The following VB sample adds the control's icons list from a BASE64 encoded string:

```
Dim s As String
With Tree1
   .BeginUpdate
   s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExn

   s = s + "Poyf5xoojKAg"
   .Images s

   .Columns.Add "Column 1"
   With .Items
      Dim h As HITEM
      h = .AddItem("Item 1")
      .CellImage(h, 0) = 1
      h = .AddItem("Item 2")
      .CellImages(h, 0) = "2,3"
   End With
   .EndUpdate
End With
```

If you run the sample you get:



The following VB sample loads images from a Microsoft Image List control:

```
Tree1.Images ImageList1.hImageList
```

The following C++ sample loads icons from a BASE64 encoded string:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_tree.BeginUpdate();
CString s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExn

s +=
"/xoAw9ZiFdxbAAGVxM5yOTzkPy+MzGRpmdx2kl2epGY1WgxmZl+Yyery2yyGHyeirGoo+(

s +=
"NbDDLO2xz5PlBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwyu0UPS/U

s +=
"kSSAAkqUU2nE20gmp5oo6JwH+eZ31EjJwB+eBn1K/AHnBWIfvwAZwACYAHsMy9clMyFeE

m_tree.Images( COleVariant( s ) );
m_tree.GetColumns().Add( "Column 1" );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItems items = m_tree.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_tree.EndUpdate();
```

The following VB.NET sample loads icons from a BASE64 encoded string:

```
Dim s As String
With AxTree1
   .BeginUpdate()
   s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExn
```

```
    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 loads icons from a BASE64 encoded string:

```
axTree1.BeginUpdate();
string s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExn

s = s + "Poyf5xoojKAg";
axTree1.Images(s);
axTree1.Columns.Add("Column 1");
int h = axTree1.Items.AddItem("Item 1");
axTree1.Items.set_CellImage(h, 0, 1 );
h = axTree1.Items.AddItem("Item 2");
axTree1.Items.set_CellImages(h, 0,"2,3");
axTree1.EndUpdate();
```

The following VFP sample loads icons from a BASE64 encoded string:

```
local s
With thisform.Tree1
    .BeginUpdate()
    s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0vmExn

    s = s +
```

```
"dr1fsFhsVjslls1ntFptVrtltt1vuFxuVzul1u13vF5vV7vl9v1/wGBnqAQEZwmCxFhYGLib/xoAw9

   s = s +
"Goo+03mM02Jzee029y2Ewum2+FnOTlGezHNx0b3/C3U258a4mP5HVvOw52s2fg2vH6ml

   s = s +
"kicAJnCbsNbDDLO2xz5PlBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwy

   s = s +
"wi/8iNPJMjSo0clvjMLuTHLkJTNCqVTSms2Tkq8jTzOcVP/BsePUocQLDQ9AJ3LtFUbR1Hqaiw

   s = s + "Poyf5xoojKAg"
   .Images(s)

   .Columns.Add("Column 1")
   With .Items
      .DefaultItem = .AddItem("Item 1")
      .CellImage(0, 0) = 1
      .DefaultItem = .AddItem("Item 2")
      .CellImages(0, 0) = "2,3"
   EndWith
   .EndUpdate()
EndWith
```

property Tree.ImageSize as Long

Retrieves or sets the size of control' icons/images/check-boxes/radio-buttons.

Type	Description
Long	A long expression that defines the size of icons the control displays.

By default, the ImageSize property is 16 (pixels). The ImageSize property specifies the size of icons being loaded using the Images method. The control's Images collection is cleared if the ImageSize property is changed, so it is recommended to set the ImageSize property before calling the Images method. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. For instance, if the ICO file to load includes different types the one closest with the size specified by ImageSize property is loaded by Images method. The ImageSize property does NOT change the height for the control's font.

The ImageSize property defines the size to display the following UI elements:

- any icon that a cell or column displays (<img>number</img> ex-html tag, CellImage, CellImages)
- check-box or radio-buttons (CellHasCheckBox, CellHasRadioButton)
- expand/collapse glyphs (HasButtons, HasButtonsCustom)
- header's sorting or drop down-filter glyphs

property Tree.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. Use the InsertControlItem property to insert an ActiveX item.

property Tree.ItemFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, ColIndex as Long, HitTestInfo as HitTestInfoEnum) as HITEM

Retrieves the item from the cursor.

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. If the return value is zero, the ColIndex may indicate the handle of the cell (inner cell).
HitTestInfo as HitTestInfoEnum	A HitTestInfoEnum expression that determines on return, the position of the cursor within the cell. If the HitTestInfo value includes the exHTCellIcon or exHTCellCaptionIcon, the four most significant bytes indicates the index of the icon from the point.
HITEM	A long expression that indicates the item's handle where the point is.

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**. Use the ColumnFromPoint property to retrieve the column from cursor. Use the SelectableItem property to specify the user can select an item.

The following VB sample prints the cell's caption from the cursor (if the control contains no inner cells. Use the SplitCell property to insert inner cells) :

```
Private Sub Tree1_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 EXTREELibCtl.HitTestInfoEnum
    ' Gets the item from (X,Y)
    h = Tree1.ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Then
        Debug.Print Tree1.Items.CellCaption(h, c) & " HT = " & hit
    End If
End Sub
```

The following VB sample displays the cell's caption from the cursor (if the control contains inner cells):

```
Private Sub Tree1_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 EXTREELibCtl.HitTestInfoEnum
    ' Gets the item from (X,Y)
    h = Tree1.ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Or Not (c = 0) Then
        Debug.Print Tree1.Items.CellCaption(h, c) & " HT = " & hit
    End If
End Sub
```

The following VB sample displays the index of icon being clicked:

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim i As HITEM, h As HitTestInfoEnum, c As Long
    With Tree1
        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 caption of the cell being double clicked (including the inner cells):

```
    EXTREELib.HitTestInfoEnum hit;
    int c = 0, h = axTree1.get_ItemFromPoint( e.x, e.y, out c, out hit );
    if ( ( h != 0 ) || ( c != 0 ) )
        MessageBox.Show( axTree1.Items.get_CellCaption( h, c ).ToString() );
```

The following VC sample displays the caption of the cell being clicked:

```
#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 OnMouseDownTree1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
    if ( ( hItem != 0 ) || ( c != 0 ) )
    {
        CItems items = m_tree.GetItems();
        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 displays the caption from the cell being clicked:

```
Private Sub AxTree1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseDownEvent) Handles AxTree1.MouseDownEvent
    With AxTree1
        Dim i As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (Not (i = 0) Or Not (c = 0)) Then
            Debug.WriteLine("Cell: " & .Items.CellCaption(i, c) & " Hit: " & hit.ToString())
        End If
    End With
End Sub
```

The following C# sample displays the caption from the cell being clicked:

```
private void axTree1_MouseDownEvent(object sender,
AxEXTREELib._ITreeEvents_MouseDownEvent e)
{
    int c = 0;
    EXTREELib.HitTestInfoEnum hit;
    int i = axTree1.get_ItemFromPoint( e.x, e.y, out c,out hit );
    if ( ( i != 0 ) || ( c != 0 ) )
    {
        string s = axTree1.Items.get_CellCaption( i,c ).ToString();
        s = "Cell: " + s + ", Hit: " + hit.ToString();
        System.Diagnostics.Debug.WriteLine( s );
    }
}
```

The following VFP sample displays the caption from the cell being clicked (the code should be in the Tree1.MouseDown event):

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

```
local c, hit
c = 0
hit = 0
with thisform.Tree1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
      wait window nowait .Items.CellCaption( 0, c ) + " " + Str( hit )
   endif
endwith
```

property Tree.Items as Items

Retrieves the control's item collection.

Type	Description
Items	An Items object that holds the control's items collection.

Use the Items property to access the Items collection. Use the Items collection to add, remove or change the control items. Use the GetItems method to get the items collection into a safe array. Use the PutItems method to load items from a safe array. Use the Columns property to access the control's Columns collection. Use the AddItem, InsertItem or InsertControlItem method to add new items to the control. Use the DataSource to add new columns and items to the control. Adding new items fails if the control has no columns.

property Tree.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 Tree.Layout as String

Saves or loads the control's layout, such as positions of the columns, scroll position, filtering values.

Type	Description
String	A String expression that specifies the control's layout.

You can use the Layout property to store the control's layout and to restore the layout later. For instance, you can save the control's Layout property to a file when the application is closing, and you can restore the control's layout when the application is loaded. The Layout property saves almost all of the control's properties that user can change at runtime (like changing the column's position by drag and drop). The Layout property does NOT save the control's data, so the Layout property should be called once you loaded the data from your database, xml or any other alternative. Once the data is loaded, you can call the Layout property to restore the View as it was saved. Before closing the application, you can call the Layout property and save the content to a file for reading next time the application is opened.

The Layout property saves/loads the following information:

- columns size and position
- current selection
- scrolling position and size
- expanded/collapsed items, if any
- sorting columns
- filtering options
- SearchColumnIndex property, indicates the column where the user can use the control's incremental searching.
- TreeColumnIndex property, which indicates the index of the column that displays the hierarchy lines.

These properties are serialized to a string and encoded in BASE64 format.

The following movies show how Layout works:

- The Layout property is used to save and restore the control's view.

Generally, the Layout property can be used to save / load the control's layout (or as it is displayed). Thought, you can benefit of this property to sort the control using one or more columns as follows:

- multiplesort="";singlesort="", removes any previously sorting
- multiplesort="C3:1", sorts ascending the column with the index 3 (and add it to the sort

bar if visible)

- singlesort="C4:2", sorts descending the column with the index 4 (it is not added to sort bar panel)
- multiplesort="C3:1";singlesort="C4:2", sorts ascending the column with the index 3 (and add it to the sort bar if visible), and sorts descending the column with the index 4. In other words, it re-sort the control by columns 3 and 4.
- multiplesort="C3:1 C5:2";singlesort="C4:2", sorts ascending the column with the index 3 (and add it to the sort bar if visible), sorts descending the column with the index 5 (and add it to the sort bar if visible), and sorts descending the column with the index 4. In other words, it re-sort the control by columns 3, 5 and 4.

The format of the Layout in non-encoded form is like follows:

```
c0.filtertype=0
c0.position=0
c0.select=0
c0.visible=1
c0.width=96
....
columns=13
collapse="0-3 5-63 80-81 83"
filterprompt=""
focus=8
focuscolumnindex=0
hasfilter=1
hscroll=0
multiplesort="C12:1 C2:2"
searchcolumnindex=3
select="39 2 13 8"
selectcolumnindex=0
singlesort="C5:2"
treecolumnindex=0
vscroll=12
vscrolloffset=0
```

property Tree.LinesAtRoot as LinesAtRootEnum

Link items at the root of the hierarchy.

Type	Description
LinesAtRootEnum	A LinesAtRootEnum expression that indicates whether the control link 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. Use the InsertControlItem property to insert an ActiveX item.

property Tree.MarkSearchColumn as Boolean

Retrieves or sets a value that indicates whether the searching column is marked or unmarked

Type	Description
Boolean	A boolean expression that indicates whether the searching column is marked or unmarked.

The control supports incremental search feature. The MarkSearchColumn property specifies whether the control highlights the searching column. Use the SearchColumnIndex property to specify the index of the searching column. The user can change the searching column by pressing the TAB ort Shift + TAB key. Use the AutoSearch property to specify whether the control enables the incremental searching feature. Use the AutoSearch property to specify the type of incremental searching the control supports within the column. Use the UseTabKey property to specify whether the control uses the TAB key.

method Tree.OLEDrag ()

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

Type	Description

Only for internal use.

property Tree.OLEDropMode as exOLEDropModeEnum

Returns or sets how a target component handles drop operations

Type	Description
exOLEDropModeEnum	An exOLEDropModeEnum expression that indicates the OLE Drag and Drop mode.

In the /NET Assembly, you have to use the AllowDrop property as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

By default, the OLEDropMode property is exOLEDropNone. The control provides the following options to define the visual effect when drag and drop items:

- Background(exDragDropBefore), Specifies the visual appearance for the drag and drop cursor before showing the items. This option can be used to apply a background to the dragging items, before painting the items. By default, the control doesn't draw any background for the items being dragged. For instance, use the Background(exDragDropBefore) = SelBackColor property to specify the same background color/skin for items being dragged as they are selected.
- Background(exDragDropAfter), Specifies the visual appearance for the drag and drop cursor after showing the items. This option can be used to apply a semi-transparent/opaque background to the dragging items, after painting the items. Use this option to apply a transparent/opaque skin, after the items are painted. For instance, using an color or an opaque skin you can show something else when dragging the items.
- Background(exDragDropListTop), Specifies the graphic feedback of the item from the drag and drop cursor if the cursor is in the top half of the row. Use this option to indicate the graphic to be displayed on the item, when the cursor is in the top half row. By default, nothing is displayed.
- Background(exDragDropListBottom), Specifies the graphic feedback of the item from the drag and drop cursor if the cursor is in the bottom half of the row. Use this option to indicate the graphic to be displayed on the item, when the cursor is in the bottom half row. By default, nothing is displayed. Use the HitTestInfoEnum.exHTBottomHalf flag to check whether the user drags the items in the top half or bottom half of the row.
- Background(exDragDropForeColor), Specifies the foreground color for the items being dragged. By default, the foreground color is black.

All options, excepts the exDragDropForeColor option accept skins. Use the Appearance.Add method to define new skins in the control.

Currently, the ExTree control supports only manual OLE Drag and Drop operation. See the OLEStartDrag and OLEDragDrop events for more details about implementing drag and drop operations into the ExTree control.

property Tree.Picture as IPictureDisp

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

Type	Description
IPictureDisp	A Picture object that's displayed on the control's background.

By default, the control has no picture associated. The control uses the PictureDisplay property to determine how the picture is displayed on the control's background. Use the PictureLevelHeader property to specify the picture on the control's levels header bar. Use the CellPicture property to assign a picture to a cell. Use the BackColor property to specify the control's background color.

property Tree.PictureDisplay as PictureDisplayEnum

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

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the picture is displayed.

By default, the PictureDisplay property is exTile. Use the PictureDisplay property specifies how the Picture is displayed on the control's background. If the control has no picture associated the PictureDisplay property has no effect. Use the CellPicture property to assign a picture to a cell. Use the BackColor property to specify the control's background color.

property Tree.PictureDisplayLevelHeader as PictureDisplayEnum

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

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the picture is displayed on the control's header.

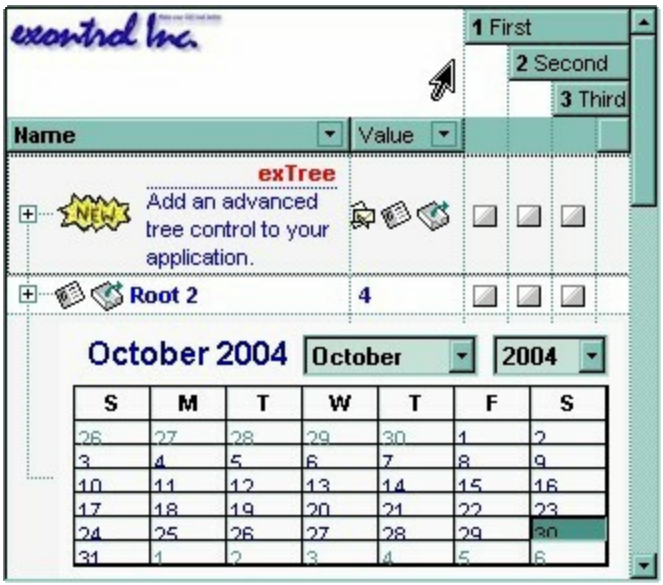
Use the PictureDisplayLevelHeader property to arrange the picture on the control's multiple levels header bar. Use the PictureLevelHeader property to load a picture on the control's header bar when it displays multiple levels. The control's header bar displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the LevelKey property to specify the control's level key.

property Tree.PictureLevelHeader as IPictureDisp

Retrieves or sets a graphic to be displayed in the control's header when multiple levels is on.

Type	Description
IPictureDisp	A Picture object being displayed on the control's header bar when multiple levels is on.

Use the PictureLevelHeader property to display a picture on the control's header bar when it displays the columns using multiple levels. Use the PictureDisplayLevelHeader property to arrange the picture on the control's multiple levels header bar. The control's header bar displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the LevelKey property to specify the control's level key. Use the Picture property to display a picture on the control's list area. Use the BackColorLevelHeader property to specify the background color for parts of the control's header bar that are not occupied by column's headers.



method Tree.PutItems (Items as Variant, [Parent as Variant])

Adds data to the control from a SafeArray containing numbers, strings, dates, or nested SafeArrays of numbers, strings, and dates, positioning them as child items of the specified parent item

Type	Description
Items as Variant	An array that control uses to fill with. The array can be one or two- dimensional. If the array is one-dimensional, the control requires one column being added before calling the PutItems method. If the Items parameter indicates a two-dimensional array, the first dimension defines the columns, while the second defines the number of items to be loaded. For instance, a(2,100) means 2 columns and 100 items. For instance: • PutItems Array("Item 1", "Item 2", "Item 3"), adds the rows at the end of the list • PutItems Array("Root", Array("Child 1", "Child 2")), adds data in a hierarchical structure, at the end of the list • PutItems rs.GetRows(), appends data from a recordset using the GetRows method of the Recordset • PutItems rs.GetRows(10), inserts the first 10 records from a Recordset using the GetRows method, at the end of the list where GetRows() method in ADO retrieves multiple records from a Recordset object and stores them in a two-dimensional array.
	Indicates one of the following: • missing, empty or 0 {number}, specifies that the data(Items) is being appended (added to the end of the list) • a long expression, that specifies the handle of the item where the array is being inserted • a string expression of of "parent;IDColumn;ParentIDColumn" format, where,

Parent as Variant	'parent' denotes the handle of the item where the data is being inserted, 'IDColumn' refers to the index of the column containing row identifiers, and 'ParentIDColumn' indicates the index of the column containing identifiers of parent rows. This way, you can insert data hierarchically using parent-id relationship. A parent-id relationship is a way of organizing data in a hierarchical structure where each element (or "child") is associated with a parent element. Please be aware that the rows of the data are inserted as they were provided by the Items parameter. Therefore, it is important that the data provided be sorted by the IDColumn so that the parent row referred to by the ParentIDColumn value is already present and can be used to insert the current row as a child of it. For instance: • PutItems Array("Item 1", "Item 2", "Item 3"), Items.ItemByIndex(2), inserts the rows as children of the item with index 2 • PutItems Array("Root", Array("Child 1", "Child 2")), Items.FirstVisibleItem, Inserts data as a hierarchical structure, placing it as a child of the first visible item • PutItems rs.GetRows(), Items.ItemByIndex(0), inserts the records from the recordset using the GetRows method of the Recordset, placing them as children of the item with index 0 • PutItems rs.GetRows(), ";0;3", inserts the records from the recordset using the GetRows method of the Recordset, utilizing parent-child relationships. The first column (index 0) contains the identifiers of the rows, while the fourth column (index 3) contains the keys of the parent rows. where GetRows() method in ADO retrieves multiple records from a Recordset object and stores them in a two-dimensional array.

The PutItems method loads items from a safe array. The PutItems method may raise one of the following exceptions:

- The array dimension exceeds 2 (In simpler terms, a two-dimensional array (or 2D array) is like a table with rows and columns. If an array exceeds 2 dimensions, it means it has three or more dimensions, such as a 3D array (which can be thought of as a collection of tables) or even higher dimensions) You need to provide a one-dimensional or two-dimensional array
- The number of columns does not match the array size (either the control has no columns or the number of columns is too small). You need to add more columns (Add property).
- The element type of the array is not valid (the type of the array is either unknown or not supported) You need to provide a valid type, which must be one of the following: Variant, String, Integer, Long, Double, Float, or Date.

The PutItems method performs:

1. **Insertion Order**: The data is inserted into the system in the same order as it is provided by the Items parameter. This means that the sequence of rows in the Items parameter directly affects how the data is inserted.
2. **Sorting Requirement**: To ensure correct insertion, it's crucial that the data is sorted by the IDColumn (when the Parent parameter is of "parent;IDColumn;ParentIDColumn" format). This sorting ensures that parent rows are inserted before their corresponding child rows.
3. **Parent-Child Relationship**: The sorting ensures that when a row refers to a parent row using the ParentIDColumn value (when the Parent parameter is of "parent;IDColumn;ParentIDColumn" format). The parent row is already present in the control. This allows the current row to be inserted as a child of the parent row without encountering errors or inconsistencies.

In essence, by sorting the data appropriately, you establish a clear hierarchy where parent rows are inserted before child rows, maintaining the integrity of the parent-child relationships within the dataset.

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

EmployeeID	EmployeeName	DepartmentID	ParentID
1	John	101	
2	Alice	102	1
3	Bob	101	1
4	Sarah	102	1
5	Emma	101	2
6	Mike	102	2

Each row represents an employee.

- EmployeeID uniquely identifies each employee (represents the column with the index 0)
- EmployeeName denotes the name of the employee (represents the column with the index 1)
- DepartmentID indicates the department to which the employee belongs (represents the column with the index 2)
- ParentID establishes the relationship between employees (represents the column with the index 3), where it references the EmployeeID of the parent employee. An empty value indicates the absence of a parent, typically representing the head of the department.

Having this data organized into a two-dimensional array, the statement PutItems d loads it as a flat table:

EmployeeID	EmployeeName	DepartmentID	ParentID
1	John	101	
2	Alice	102	1
3	Bob	101	1
4	Sarah	102	1
5	Emma	101	2
6	Mike	102	2

whereas PutItems d, ";0;3" loads it as a tree structure:

EmployeeID	EmployeeName	DepartmentID	ParentID
1	⊟ John	101	
2	⊟ Alice	102	1
5	Emma	101	2
6	Mike	102	2
3	Bob	101	1
4	Sarah	102	1

where d is an array as defined next:

```
Dim d(3, 5) As Variant
d(0, 0) = "1": d(1, 0) = "John": d(2, 0) = "101": d(3, 0) = ""
d(0, 1) = "2": d(1, 1) = "Alice": d(2, 1) = "102": d(3, 1) = "1"
d(0, 2) = "3": d(1, 2) = "Bob": d(2, 2) = "101": d(3, 2) = "1"
d(0, 3) = "4": d(1, 3) = "Sarah": d(2, 3) = "102": d(3, 3) = "1"
d(0, 4) = "5": d(1, 4) = "Emma": d(2, 4) = "101": d(3, 4) = "2"
d(0, 5) = "6": d(1, 5) = "Mike": d(2, 5) = "102": d(3, 5) = "2"
```

Use the GetItems method to get a safe array with the items in the control. The PutItems method fires AddItem event for each item added to Items collection. Use the Items property to access the items collection. 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

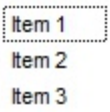
the value of a formula.

The following VB6 sample loads a flat array to a single column control (and shows as in the following picture):

```
With Tree1
   .BeginUpdate
   .Columns.Add "Column 1"
   .PutItems Array("Item 1", "Item 2", "Item 3")
   .EndUpdate
End With
```

or similar for /NET Assembly version:

```
 With Extree1
   .BeginUpdate()
   .Columns.Add("Column 1")
   .PutItems(New String() {"Item 1", "Item 2", "Item 3"})
   .EndUpdate()
End With
```



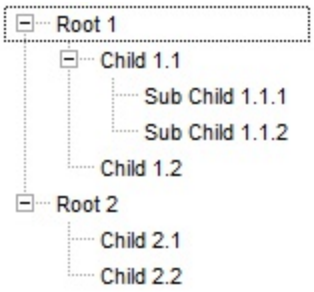
The following VB6 sample loads a hierarchy to a single column control (and shows as in the following picture):

```
With Tree1
   .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 Extree1
   .BeginUpdate()
```

```
    .LinesAtRoot = exontrol.EXTREELib.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 VB6 sample loads a list of items, in a three columns control (as shown in the following picture):

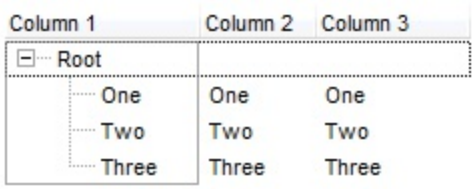
```
Dim v(2, 2) As String
v(0, 0) = "One": v(0, 1) = "Two": v(0, 2) = "Three"
v(1, 0) = "One": v(1, 1) = "Two": v(1, 2) = "Three"
v(2, 0) = "One": v(2, 1) = "Two": v(2, 2) = "Three"

With Tree1
    .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 VB6 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 Tree1
   .BeginUpdate
   .Columns.Add "Column 1"
   .Columns.Add "Column 2"
   .Columns.Add "Column 3"

   .Items.AddItem "Root"

   .PutItems v, .Items.FirstVisibleItem
   .EndUpdate
End With
```



The following VB sample loads the collection of records from an ADO recordset:

```
Dim rs As Object
Const dwProvider = "Microsoft.Jet.OLEDB.4.0" ' OLE Data provider
Const nCursorType = 3 ' adOpenStatic
Const nLockType = 3 ' adLockOptimistic
Const nOptions = 2 ' adCmdTable
Const strDatabase = "D:\Program Files\Microsoft Visual Studio\VB98\NWIND.MDB"

'Creates an recordset and opens the "Employees" table, from NWIND database
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Employees", "Provider=" & dwProvider & ";Data Source= " & strDatabase,
nCursorType, nLockType, nOptions
With Tree1
   .BeginUpdate

      .ColumnAutoResize = False
```

```
      .MarkSearchColumn = False
      .DrawTreeLines = True
      ' Adds a column for each field found
      With .Columns
         Dim f As Object
         For Each f In rs.Fields
            .Add f.Name
         Next
      End With

      ' Loads the collection of records
      .PutItems rs.GetRows()
   .EndUpdate
End With
```

The following C++ sample loads records from an ADO recordset, using the PutItems method:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"



#pragma warning( disable : 4146 )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
using namespace ADODB;



_RecordsetPtr spRecordset;
if ( SUCCEEDED( spRecordset.CreateInstance( "ADODB.Recordset") ) )
{
   // Builds the connection string.
   CString strTableName = "Employees", strConnection =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
   CString strPath = "D:\\Program Files\\Microsoft Visual Studio\\VB98\\NWIND.MDB";
   strConnection += strPath;
   try
   {
```

```
      // Loads the table
      if ( SUCCEEDED( spRecordset->Open(_variant_t( (LPCTSTR)strTableName ),
_variant_t((LPCTSTR)strConnection), adOpenStatic, adLockPessimistic, NULL ) ) )
      {
         m_tree.BeginUpdate();
         m_tree.SetColumnAutoResize( FALSE );
         CColumns columns = m_tree.GetColumns();
         for ( long i = 0; i < spRecordset->Fields->Count; i++ )
            columns.Add( spRecordset->Fields->GetItem(i)->Name );
         COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
         m_tree.PutItems( &spRecordset->GetRows(-1), vtMissing );
         m_tree.EndUpdate();
      }
   }
   catch ( _com_error& e )
   {
      AfxMessageBox( e.Description() );
   }
}
```

The sample uses the #import statement to import ADODB recordset's type library. The sample enumerates the fields in the recordset and adds a new column for each field found. Also, the sample uses the GetRows method of the ADODB recordset to retrieves multiple records of a Recordset object into a safe array. Please consult the ADODB documentation for the GetRows property specification.

property Tree.RadioImage(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 RadioImage and CheckImage properties to define the icons used for radio and check box cells. The RadioImage property defines the index of the icon being used by radio buttons. Use the CellHasRadioButton property to assign a radio button to a cell. Use the CellHasCheckBox property to assign a checkbox to a cell. Use the CellImage or CellImages property to assign one or multiple icons to a cell. Use the CellPicture property to assign a picture to a cell. Use the CellStateChanged event to notify your application when the cell's state is changed. Use the PartialCheck property to allow partial check feature within the column. Use the Images method to insert icons at runtime. The following samples require a control with icons, else nothing will be changed. The ImageSize property defines the size (width/height) of the control's radio buttons.

The following VB sample changes the default icon for the cells of radio type:

```
Tree1.RadioImage(True) = 1      ' Sets the icon for cells of radio type that are checked
Tree1.RadioImage(False) = 2     ' Sets the icon for cells of radio type that are
unchecked
```

The Tree1.RadioImage(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_tree.SetRadioImage( TRUE, 1 );
m_tree.SetRadioImage( FALSE, 2 );
```

The following VB.NET sample changes the default icon for the cells of radio type:

```
With AxTree1
   .set_RadioImage(True, 1)
   .set_RadioImage(False, 2)
End With
```

The following C# sample changes the default icon for the cells of radio type:

```
axTree1.set_RadioImage(true, 1);
axTree1.set_RadioImage(false, 2);
```

The following VFP sample changes the default icon for the cells of radio type:

```
with thisform.Tree1
   local sT, sCR
   sCR = chr(13) + chr(10)
   sT = "RadioImage(True) = 1"+ sCR
   sT = sT + "RadioImage(False) = 2"+ sCR
   .Template = sT
endwith
```

The VFP considers the RadioImage 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.

property Tree.RClickSelect as Boolean

Retrieves or sets a value that indicates whether an item is selected using right mouse button.

Type	Description
Boolean	A boolean expression that indicates whether an item is selected using the right mouse button.

Use the RClickSelect property to allow users select items using the right click. By default, the RClickSelect property is False. The control fires the SelectionChanged event when user selects an item. Use the SelectItem property to select programmatically select an item. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the FocusItem property to get the focused item. Use the ItemFromPoint property to retrieve an item from the point.

method Tree.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:

```
Tree1.Refresh
```

The following C++ sample calls the Refresh method:

```
m_tree.Refresh();
```

The following VB.NET sample calls the Refresh method:

```
AxTree1.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:

```
axTree1.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.Tree1.Object.Refresh()
```

method Tree.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 includes no descendents). The UnselectAll method unselects all items in the list.

method Tree.ReplaceIcon ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the control's image list.

Type	Description
Icon as Variant	A Variant expression that specifies the icon to add or insert, as one of the following options: • a long expression that specifies the handle of the icon (HICON) • a string expression that indicates the path to the picture file • a string expression that defines the picture's content encoded as BASE64 strings using the eXImages tool • a Picture reference, which is an object that holds image data. It is often used in controls like PictureBox, Image, or in custom controls (e.g., IPicture, IPictureDisp) If the Icon parameter is 0, it specifies that the icon at the given Index is removed. Furthermore, setting the Index parameter to -1 removes all icons. By default, if the Icon parameter is not specified or is missing, a value of 0 is used.
Index as Variant	A long expression that defines the index of the icon to insert or remove, as follows: • A zero or positive value specifies the index of the icon to insert (when Icon is non-zero) or to remove (when the Icon parameter is zero) • A negative value clears all icons when the Icon parameter is zero By default, if the Index parameter is not specified or is missing, a value of -1 is used.

Return	Description
Long	A long expression that indicates the index of the icon in the images collection

Use the ReplaceIcon property to add, remove or replace an icon in the control's images

collection. Also, the ReplaceIcon property can clear the images collection. Use the Images method to attach a image list to the control.

The following VB sample adds a new icon to control's images list:

```
 i = ExTree1.ReplaceIcon( LoadPicture("d:\icons\help.ico").Handle), i specifies the index where the icon is added
```

The following VB sample replaces an icon into control's images list::

```
 i = ExTree1.ReplaceIcon( LoadPicture("d:\icons\help.ico").Handle, 0), i is zero, so the first icon is replaced.
```

The following VB sample removes an icon from control's images list:

```
 ExTree1.ReplaceIcon 0,  i, where i specifies the index of icon removed.
```

The following VB clears the control's icons collection:

```
 ExTree1.ReplaceIcon 0,  -1
```

property Tree.RightToLeft as Boolean

Indicates whether the component should draw right-to-left for RTL languages.

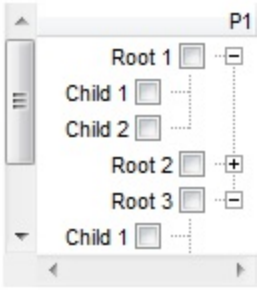
Type	Description
Boolean	A boolean expression that specifies whether the control is drawn from right to left or from left to right.

By default, the RightToLeft property is False. The RightToLeft gets or sets a value indicating whether control's elements are aligned to right or left. The RightTolLeft property affects all columns, and future columns being added.

Changing the RightToLeft property on True does the following:

- displays the vertical scroll bar on the left side of the control (Scrollbars property)
- flips the order of the columns (Position property)
- change the column's alignment to right, if the column is not centered (Alignment property, HeaderAlignment property, HeaderImageAlignment property)
- reverse the order of the drawing parts in the cells (Def(exCellDrawPartsOrder) property to "caption,picture,icons,icon,check")
- aligns the locked columns to the right (CountLockedColumns property)
- aligns the control's group-by bar / sort bar to the right (SortBarVisible property)
- the control's filter bar/prompt/close is aligned to the right (FilterBarPromptVisible property)

The following screen shot shows how the control looks if the RightToLeft property is True:

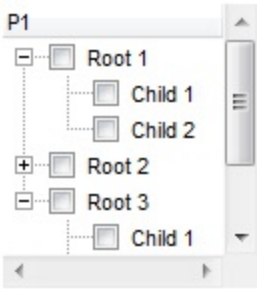


(By default) Changing the RightToLeft property on False does the following:

- displays the vertical scroll bar on the right side of the control (Scrollbars property)
- flips the order of the columns (Position property)
- change the column's alignment to left, if the column is not centered (Alignment property, HeaderAlignment property, HeaderImageAlignment property)
- reverse the order of the drawing parts in the cells (Def(exCellDrawPartsOrder) property to "check,icon,icons,picture,caption")
- aligns the locked columns to the left (CountLockedColumns property)
- aligns the control's group-by bar / sort bar to the left (SortBarVisible property)

- the control's filter bar/prompt/close is aligned to the left (FilterBarPromptVisible property)

The following screen shot shows how the control looks if the RightToLeft property is False:



The following VB sample shows how to change the order of the columns from right to left

```
With Tree1
   .BeginUpdate
   .ScrollBars = exDisableBoth
   .LinesAtRoot = exLinesAtRoot
   With .Columns.Add("P1")
      .Def(exCellHasCheckBox) = True
      .PartialCheck = True
   End With
   With .Items
      h = .AddItem("Root")
      .InsertItem h,0,"Child 1"
      .InsertItem h,0,"Child 2"
      .ExpandItem(h) = True
   End With
   .RightToLeft = True
   .EndUpdate
End With
```

The following VB.NET sample shows how to change the order of the columns from right to left

```
Dim h
With AxTree1
   .BeginUpdate
   .ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth
   .LinesAtRoot = EXTREELib.LinesAtRootEnum.exLinesAtRoot
```

```
    With .Columns.Add("P1")
        .Def(EXTREELib.DefColumnEnum.exCellHasCheckBox) = True
        .PartialCheck = True
    End With
    With .Items
        h = .AddItem("Root")
        .InsertItem h,0,"Child 1"
        .InsertItem h,0,"Child 2"
        .ExpandItem(h) = True
    End With
    .RightToLeft = True
    .EndUpdate
End With
```

The following C++ sample shows how to change the order of the columns from right to left

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

    #import <ExTree.dll>
    using namespace EXTREELib;
*/
EXTREELib::ITreePtr spTree1 = GetDlgItem(IDC_TREE1)->GetControlUnknown();
spTree1->BeginUpdate();
spTree1->PutScrollBars(EXTREELib::exDisableBoth);
spTree1->PutLinesAtRoot(EXTREELib::exLinesAtRoot);
EXTREELib::IColumnPtr var_Column = ((EXTREELib::IColumnPtr)(spTree1->GetColumns()-
>Add(L"P1")));
    var_Column->PutDef(EXTREELib::exCellHasCheckBox,VARIANT_TRUE);
    var_Column->PutPartialCheck(VARIANT_TRUE);
EXTREELib::IItemsPtr var_Items = spTree1->GetItems();
    long h = var_Items->AddItem("Root");
    var_Items->InsertItem(h,long(0),"Child 1");
    var_Items->InsertItem(h,long(0),"Child 2");
    var_Items->PutExpandItem(h,VARIANT_TRUE);
spTree1->PutRightToLeft(VARIANT_TRUE);
```

```
spTree1->EndUpdate();
```

The following C# sample shows how to change the order of the columns from right to left

```
axTree1.BeginUpdate();
axTree1.ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth;
axTree1.LinesAtRoot = EXTREELib.LinesAtRootEnum.exLinesAtRoot;
EXTREELib.Column var_Column = (axTree1.Columns.Add("P1") as EXTREELib.Column);
   var_Column.set_Def(EXTREELib.DefColumnEnum.exCellHasCheckBox,true);
   var_Column.PartialCheck = true;
EXTREELib.Items var_Items = axTree1.Items;
   int h = var_Items.AddItem("Root");
   var_Items.InsertItem(h,0,"Child 1");
   var_Items.InsertItem(h,0,"Child 2");
   var_Items.set_ExpandItem(h,true);
axTree1.RightToLeft = true;
axTree1.EndUpdate();
```

The following VFP sample shows how to change the order of the columns from right to left

```
with thisform.Tree1
   .BeginUpdate
   .ScrollBars = 15
   .LinesAtRoot = -1
   with .Columns.Add("P1")
      .Def(0) = .T.
      .PartialCheck = .T.
   endwith
   with .Items
      h = .AddItem("Root")
      .InsertItem(h,0,"Child 1")
      .InsertItem(h,0,"Child 2")
      .DefaultItem = h
      .ExpandItem(0) = .T.
   endwith
   .RightToLeft = .T.
   .EndUpdate
endwith
```

The following Delphi sample shows how to change the order of the columns from right to left

```
with AxTree1 do
begin
   BeginUpdate();
   ScrollBars := EXTREELib.ScrollBarsEnum.exDisableBoth;
   LinesAtRoot := EXTREELib.LinesAtRootEnum.exLinesAtRoot;
   with (Columns.Add('P1') as EXTREELib.Column) do
   begin
      Def[EXTREELib.DefColumnEnum.exCellHasCheckBox] := TObject(True);
      PartialCheck := True;
   end;
   with Items do
   begin
      h := AddItem('Root');
      InsertItem(h,TObject(0),'Child 1');
      InsertItem(h,TObject(0),'Child 2');
      ExpandItem[h] := True;
   end;
   RightToLeft := True;
   EndUpdate();
end
```

method Tree.Scroll (Type as ScrollEnum, [ScrollTo as Variant])

Scrolls the control's content.

Type	Description
Type as ScrollEnum	A ScrollEnum expression that indicates type of scrolling being performed.
ScrollTo as Variant	A long expression that indicates the position where the control is scrolled when Type is exScrollVTo or exScrollHTo. If the ScrollTo parameter is missing, 0 value is used.

Use the Scroll method to scroll the control's content by code. Use the EnsureVisibleItem method to ensure that a specified item fits the control's client area. Use the ScrollPos property to get the control's scroll position. Use the EnsureVisibleColumn method to ensure that a specified column fits the control's client area. If the Type parameter is exScrollLeft, exScrollRight or exScrollHTo the Scroll method scrolls horizontally the control's content pixel by pixel, if the ContinueColumnScroll property is False, else the Scroll method scrolls horizontally the control's content column by column.

The following VB sample scrolls the control's content to the first item (scrolls to the top):

```
Tree1.Scroll exScrollVTo, 0
```

The following C++ sample scrolls the control's content to the top:

```
m_tree.Scroll( 2 /*exScrollVTo*/, COleVariant( (long)0 ) );
```

The following C# sample scrolls the control's content to the top:

```
axTree1.Scroll(EXTREELib.ScrollEnum.exScrollVTo, 0);
```

The following VB.NET sample scrolls the control's content to the top:

```
AxTree1.Scroll(EXTREELib.ScrollEnum.exScrollVTo, 0)
```

The following VFP sample scrolls the control's content to the top:

```
with thisform.Tree1
   .Scroll( 2, 0 ) && exScrollVTo
endwith
```

property Tree.ScrollBars as ScrollBarsEnum

Returns or sets a value that determines whether the control has horizontal and/or vertical scroll bars.

Type	Description
ScrollBarsEnum	A ScrollBarsEnum expression that identifies which scroll bars are visible.

Use the ScrollBars property to disable the control's scroll bars. By default, the ScrollBars property is exBoth, so both scroll bars are used if necessarily. For instance, if the ScrollBars property is exNone the control displays no scroll bars. Use the ScrollPos property to get the control's scroll position. Use the EnsureVisibleItem method to ensure that an item fits the control's client area. Use the EnsureVisibleColumn method to ensure that a specified column fits the control's client area. Use the Scroll method to scroll programmatically the control. Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part. 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.

property Tree.ScrollButtonHeight as Long

Specifies the height of the button in the vertical scrollbar.

Type	Description
Long	A long expression that defines the height of the button in the vertical scroll bar.

By default, the ScrollButtonHeight property is -1. If the ScrollButtonHeight property is -1, the control uses the default height (from the system) for the buttons in the vertical scroll bar. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollBars property to specify which scroll bar is visible or hidden in the control. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property Tree.ScrollButtonWidth as Long

Specifies the width of the button in the horizontal scrollbar.

Type	Description
Long	A long expression that defines the width of the button in the horizontal scroll bar.

By default, the ScrollButtonWidth property is -1. If the ScrollButtonWidth property is -1, the control uses the default width (from the system) for the buttons in the horizontal scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollBars property to specify which scroll bar is visible or hidden in the control. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property Tree.ScrollBySingleLine as Boolean

Retrieves or sets a value that indicates whether the control scrolls the lines to the end, item by item.

Type	Description
Boolean	A boolean expression that indicates whether the control scrolls the lines to the end, item by item.

By default, the ScrollBySingleLine property is False. We recommend to set the ScrollBySingleLine property on True if you have one of the following:

- If you have at least a cell that has CellSingleLine property on exCaptionWordWrap / exCaptionBreakWrap / False, or a column with Def(exCellSingleLine) on exCaptionWordWrap / exCaptionBreakWrap / False
- If your control contains at least an item that hosts an ActiveX control. See InsertControlItem property.
- If the control displays items with different height. Use the ItemHeight property to specify the item's height.

Use the EnsureVisibleItem property to ensure that an item fits the control's client area. Use the ScrollBars property to hide the control's scroll bars. Use the Scroll method to programmatically scroll the control's content. 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 Tree.ScrollFont (ScrollBar as ScrollBarEnum) as IFontDisp

Retrieves or sets the scrollbar's font.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical or the horizontal scroll bar.
IFontDisp	A Font object

Use the ScrollFont property to specify the font in the control's scroll bar. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrollBars property to specify the visible scrollbars in the control. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar.

property Tree.ScrollHeight as Long

Specifies the height of the horizontal scrollbar.

Type	Description
Long	A long expression that defines the height of the horizontal scroll bar.

By default, the ScrollHeight property is -1. If the ScrollHeight property is -1, the control uses the default height of the horizontal scroll bar from the system. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollBars property to specify which scroll bar is visible or hidden in the control. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property Tree.ScrollOrderParts(ScrollBar as ScrollBarEnum) as String

Specifies the order of the buttons in the scroll bar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the order of buttons is displayed.
String	A String expression that indicates the order of the buttons in the scroll bar. The list includes expressions like l, l1, ..., l5, t, r, r1, ..., r6 separated by comma, each expression indicating a part of the scroll bar, and its position indicating the displaying order.

Use the ScrollOrderParts to customize the order of the buttons in the scroll bar. By default, the ScrollOrderParts property is empty. If the ScrollOrderParts property is empty the default order of the buttons in the scroll bar are displayed like follows:

so, the order of the parts is: l1, l2, l3, l4, l5, l, t, r, r1, r2, r3, r4, r5 and r6. Use the ScrollPartVisible to specify whether a button in the scrollbar is visible or hidden. Use the ScrollPartEnable property to enable or disable a button in the scroll bar. Use the ScrollPartCaption property to assign a caption to a button in the scroll bar.

Use the ScrollOrderParts property to change the order of the buttons in the scroll bar. For instance, "l,r,t,l1,r1" puts the left and right buttons to the left of the thumb area, and the l1 and r1 buttons right after the thumb area. If the parts are not specified in the ScrollOrderParts property, automatically they are added to the end.

The list of supported literals in the ScrollOrderParts property is:

- **l** for exLeftBPart, (<) The left or top button.
- **l1** for exLeftB1Part, (L1) The first additional button, in the left or top area.
- **l2** for exLeftB2Part, (L2) The second additional button, in the left or top area.
- **l3** for exLeftB3Part, (L3) The third additional button, in the left or top area.
- **l4** for exLeftB4Part, (L4) The forth additional button, in the left or top area.
- **l5** for exLeftB5Part, (L5) The fifth additional button, in the left or top area.
- **t** for exLowerBackPart, exThumbPart and exUpperBackPart, The union between the exLowerBackPart and the exUpperBackPart parts.
- **r** for exRightBPart, (>) The right or down button.
- **r1** for exRightB1Part, (R1) The first additional button in the right or down side.

- **r2** for exRightB2Part, (R2) The second additional button in the right or down side.
- **r3** for exRightB3Part, (R3) The third additional button in the right or down side.
- **r4** for exRightB4Part, (R4) The forth additional button in the right or down side.
- **r5** for exRightB5Part, (R5) The fifth additional button in the right or down side.
- **r6** for exRightB6Part, (R6) The sixth additional button in the right or down side.

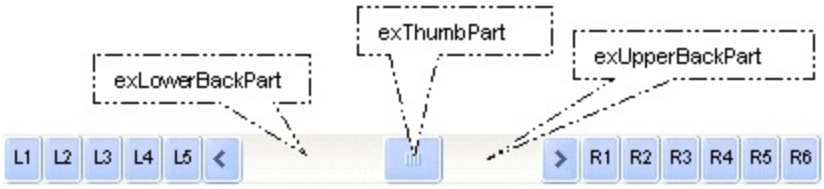
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property Tree.ScrollPartCaption(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as String

Specifies the caption being displayed on the specified scroll part.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the caption is displayed.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll where the text is displated
String	A String expression that specifies the caption being displayed on the part of the scroll bar.

Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrollBars property to specify the visible scrollbars in the control. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar. Use the ScrollFont property to specify the font in the control's scroll bar. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.



By default, the following parts are shown:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

The following VB sample adds up and down additional buttons to the control's vertical scroll bar :

```
With Tree1
   .BeginUpdate
      .ScrollBars = exDisableBoth
      .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
      .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
      .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
   .EndUpdate
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxTree1
   .BeginUpdate()
   .ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth
   .set_ScrollPartVisible(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part Or EXTREELib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.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 :

```
axTree1.BeginUpdate();
axTree1.ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth;
axTree1.set_ScrollPartVisible(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part | EXTREELib.ScrollPartEnum.exRightB1Part, true);
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axTree1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_tree.BeginUpdate();
m_tree.SetScrollBars( 15 /*exDisableBoth*/ );
m_tree.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_tree.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_tree.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2")
);
m_tree.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Tree1
   .BeginUpdate
      .ScrollBars = 15
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWith
```

```
*** ActiveX Control Event ***
LPARAMETERS scrollpart

wait window nowait ltrim(str(scrollpart))
```

property Tree.ScrollPartCaptionAlignment(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as AlignmentEnum

Specifies the alignment of the caption in the part of the scroll bar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the caption is displayed.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll where the text is displayed
AlignmentEnum	An AlignmentEnum expression that specifies the alignment of the caption in the part of the scrollbar.

The ScrollPartCaptionAlignment property specifies the alignment of the caption in the part of the scroll bar. By default, the caption is centered. Use the ScrolPartCaption property to specify the caption being displayed on specified part of the scroll bar. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar.

The following VB sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
With Tree1
   .ScrollPartCaption(exHScroll,exLowerBackPart) = "left"
   .ScrollPartCaptionAlignment(exHScroll,exLowerBackPart) = LeftAlignment
   .ScrollPartCaption(exHScroll,exUpperBackPart) = "right"
   .ScrollPartCaptionAlignment(exHScroll,exUpperBackPart) = RightAlignment
   .ColumnAutoResize = False
   .Columns.Add 1
   .Columns.Add 2
   .Columns.Add 3
   .Columns.Add 4
End With
```

The following VB.NET sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
With AxTree1
```

```
.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.ScrollPartEnum.exLow


.set_ScrollPartCaptionAlignment(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.ScrollPartEr


.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.ScrollPartEnum.exUpp


.set_ScrollPartCaptionAlignment(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.ScrollPartEr

    .ColumnAutoResize = False
    .Columns.Add 1
    .Columns.Add 2
    .Columns.Add 3
    .Columns.Add 4
End With
```

The following C# sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.ScrollPartEnum


axTree1.set_ScrollPartCaptionAlignment(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.Scrc


axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.ScrollPartEnum


axTree1.set_ScrollPartCaptionAlignment(EXTREELib.ScrollBarEnum.exHScroll,EXTREELib.Scrc


axTree1.ColumnAutoResize = false;
axTree1.Columns.Add(1.ToString());
axTree1.Columns.Add(2.ToString());
axTree1.Columns.Add(3.ToString());
axTree1.Columns.Add(4.ToString());
```

The following C++ sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's

horizontal scroll bar:

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

   #import "ExTree.dll"
   using namespace EXTREELib;
*/
EXTREELib::ITreePtr spTree1 = GetDlgItem(IDC_TREE1)->GetControlUnknown();
spTree1->PutScrollPartCaption(EXTREELib::exHScroll,EXTREELib::exLowerBackPart,L"left");
spTree1-
>PutScrollPartCaptionAlignment(EXTREELib::exHScroll,EXTREELib::exLowerBackPart,EXTREE

spTree1-
>PutScrollPartCaption(EXTREELib::exHScroll,EXTREELib::exUpperBackPart,L"right");
spTree1-
>PutScrollPartCaptionAlignment(EXTREELib::exHScroll,EXTREELib::exUpperBackPart,EXTREE

spTree1->PutColumnAutoResize(VARIANT_FALSE);
spTree1->GetColumns()->Add(L"1");
spTree1->GetColumns()->Add(L"2");
spTree1->GetColumns()->Add(L"3");
spTree1->GetColumns()->Add(L"4");
```

The following VFP sample displays "left" aligned to the left on the lower part of the control's horizontal scroll bar, and "right" aligned to the right on the upper part of the control's horizontal scroll bar:

```
with thisform.Tree1
   .ScrollPartCaption(1,512) = "left"
   .ScrollPartCaptionAlignment(1,512) = 0
   .ScrollPartCaption(1,128) = "right"
   .ScrollPartCaptionAlignment(1,128) = 2
   .ColumnAutoResize = .F.
   .Columns.Add(1)
   .Columns.Add(2)
   .Columns.Add(3)
```

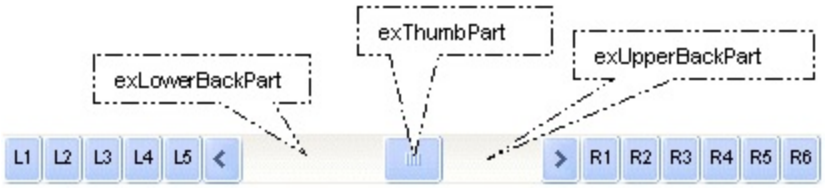
```
    .Columns.Add(4)
endwith
```

property Tree.ScrollPartEnable(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

Indicates whether the specified scroll part is enabled or disabled.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the part is enabled or disabled.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll bar being enabled or disabled.
Boolean	A Boolean expression that specifies whether the scrollbar's part is enabled or disabled.

By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrollBars property to specify the visible scrollbars in the control. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.

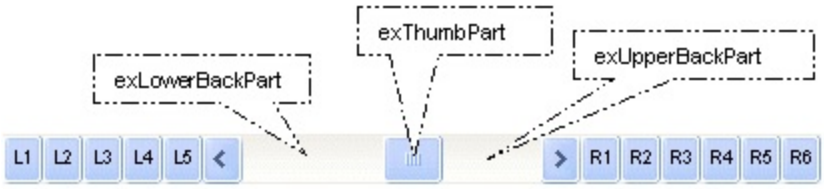


property Tree.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

Indicates whether the specified scroll part is visible or hidden.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the part is visible or hidden.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll bar being visible
Boolean	A Boolean expression that specifies whether the scrollbar's part is visible or hidden.

Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the ScrollPartEnable property is automatically called, so the parts becomes enabled. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrollBars property to specify the visible scrollbars in the control. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. The control fires the ScrollButtonClick event when the user clicks a part of the scroll bar. Use the Background property to change the visual appearance for any part in the control's scroll bar. Use the ScrollOrderParts property to customize the order of the buttons in the scroll bar.



By default, the following parts are shown:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

The following VB sample adds up and down additional buttons to the control's vertical scroll bar :

```
With Tree1
   .BeginUpdate
      .ScrollBars = exDisableBoth
      .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
      .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
      .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
   .EndUpdate
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxTree1
   .BeginUpdate()
   .ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth
   .set_ScrollPartVisible(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part Or EXTREELib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.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 :

```
axTree1.BeginUpdate();
axTree1.ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth;
axTree1.set_ScrollPartVisible(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part | EXTREELib.ScrollPartEnum.exRightB1Part, true);
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axTree1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_tree.BeginUpdate();
m_tree.SetScrollBars( 15 /*exDisableBoth*/ );
m_tree.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_tree.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_tree.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2")
);
m_tree.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Tree1
   .BeginUpdate
      .ScrollBars = 15
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWith
```

```
*** ActiveX Control Event ***
LPARAMETERS scrollpart

wait window nowait ltrim(str(scrollpart))
```

property Tree.ScrollPos(Vertical as Boolean) as Long

Specifies the vertical/horizontal scroll position.

Type	Description
Vertical as Boolean	A boolean expression that specifies the scrollbar being requested. True indicates the Vertical scroll bar, False indicates the Horizontal scroll bar.
Long	A long expression that defines the scroll bar position.

Use the ScrollPos property to change programmatically the position of the control's scroll bar. Use the ScrollPos property to get the horizontal or vertical scroll position.Use the ScrollBars property to define the control's scroll bars. Use the Scroll method to scroll programmatically the control's content. The control fires the OffsetChanged event when the control's scroll position is changed. Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar.

property Tree.ScrollThumbSize(ScrollBar as ScrollBarEnum) as Long

Specifies the size of the thumb in the scrollbar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical or the horizontal scroll bar.
Long	A long expression that defines the size of the scrollbar's thumb.

Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb. By default, the ScrollThumbSize property is -1, that makes the control computes automatically the size of the thumb based on the scrollbar's range. If case, use the fixed size for your thumb when you change its visual appearance using the Background(exVSThumb) or Background(exHSThumb) property. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar.

property Tree.ScrollToolTip(ScrollBar as ScrollBarEnum) as String

Specifies the tooltip being shown when the user moves the scroll box.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical scroll bar or the horizontal scroll bar.
String	A string expression being shown when the user clicks and moves the scrollbar's thumb.

Use the ScrollToolTip property to specify whether the control displays a tooltip when the user clicks and moves the scrollbar's thumb. By default, the ScrollToolTip property is empty. If the ScrollToolTip property is empty, the tooltip is not shown when the user clicks and moves the thumb of the scroll bar. The OffsetChanged event notifies your application that the user changes the scroll position. Use the SortPartVisible property to specify the parts being visible in the control's scroll bar. Use the ScrollBars property to specify the visible scrollbars in the control.

The following VB sample displays a tooltip when the user clicks and moves the thumb in the control's scroll bar:

```
Private Sub Tree1_OffsetChanged(ByVal Horizontal As Boolean, ByVal NewVal As Long)
  If (Not Horizontal) Then
    Tree1.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 AxTree1_OffsetChanged(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OffsetChangedEvent) Handles AxTree1.OffsetChanged
  If (Not e.horizontal) Then
    AxTree1.set_ScrollToolTip(EXTREELib.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 OnOffsetChangedTree1(BOOL Horizontal, long NewVal)
{
   if ( !Horizontal )
   {
      CString strFormat;
      strFormat.Format( _T("%i"), NewVal );
      m_tree.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 axTree1_OffsetChanged(object sender,
AxEXTREELib._ITreeEvents_OffsetChangedEvent e)
{
   if ( !e.horizontal )
      axTree1.set_ScrollToolTip(EXTREELib.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.Tree1.ScrollToolTip(0) = "Record " + ltrim(str(newval))
EndIf
```

property Tree.ScrollWidth as Long

Specifies the width of the vertical scrollbar.

Type	Description
Long	A long expression that defines the width of the vertical scroll bar.

By default, the ScrollWidth property is -1. If the ScrollWidth property is -1, the control uses the default width of the vertical scroll bar from the system. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the ScrollBars property to specify which scroll bar is visible or hidden in the control. Use the ScrollButtonWidth property to specify the width of the buttons in the horizontal scroll bar. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the ScrollButtonHeight property to specify the height of the buttons in the vertical scroll bar. Use the ScrollPartVisible property to specify the visible parts in the control's scroll bar. Use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb.

property Tree.SearchColumnIndex as Long

Retrieves or sets a value indicating the column's index that is used for auto search feature.

Type	Description
Long	A long expression indicating the column's index that is used for auto search feature.

The SearchColumnIndex property indicates the index of the column being used by the control's incremental search feature. The user changes the searching column if he presses TAB or Shift + TAB. Use the UseTabKey property to specify whether the control uses the TAB key. Use the AutoSearch property to specify whether the control enables the incremental searching feature. Use the AutoSearch property to specify the type of incremental searching the control supports within the column. Use the MarkSearchColumn property to hide the rectangle around the searching column.

property Tree.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 used for selected items. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. The control fires the SelectionChanged event when user changes the selection. Use the SelectableItem property to specify the user can select an item. 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 Tree1
   With .VisualAppearance
      .Add &H23, App.Path + "\selected.ebn"
   End With
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = .SelBackColor Or &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_tree.GetVisualAppearance().Add( 0x23,
COleVariant(_T("D:\\Temp\\ExTree_Help\\selected.ebn")) );
m_tree.SetSelBackColor( m_tree.GetSelBackColor() | 0x23000000 );
m_tree.SetSelForeColor( 0 );
```

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

```
With AxTree1
   With .VisualAppearance
      .Add(&H23, "D:\Temp\ExTree_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):

```
axTree1.VisualAppearance.Add(0x23, "D:\\Temp\\ExTree_Help\\selected.ebn");
axTree1.Template = "SelBackColor = 587202560";
```

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

```
With thisform.Tree1
   With .VisualAppearance
      .Add(35, "D:\Temp\ExTree_Help\selected.ebn")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = .SelBackColor + 587202560
EndWith
```

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

How do I assign a new look for the selected item?

The component supports skinning parts of the control, including the selected item. Shortly, the idea is that identifier of the skin being added to the Appearance collection is stored in the first significant byte of property of the color type. In our case, we know that the SelBackColor property changes the background color for the selected item. This is what we need to change. In other words, we need to change the visual appearance for the selected item, and that means changing the background color of the selected item. So, the following code (blue code) changes the appearance for the selected item:

```
With Tree1
   .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 Tree1
   .VisualAppearance.Add &H34, App.Path + "\aqua.ebn"
   .SelBackColor = &H34
End With
```

This code (red code) DOESN'T use any skin, because the 34 value is not stored in the higher byte of the color value. The sample just changes the background color for the selected item to some black color (RGB(0,0,34)). So, please pay attention when you want to use a skin and when to use a color. Simple, if you are calling &H**34**000000, you have 34 followed by 6 (six) zeros, and that means the first significant byte of the color expression. Now, back to the problem. The next step is how we are creating skins? or EBN files? The Exontrol's exbutton component includes a builder tool that saves skins to EBN files. So, if you want to create new skin files, you need to download and install the exbutton component from our web site. Once that the exbutton component is installed, please follow the steps.

Let's say that we have a BMP file, that we want to stretch on the selected item's background.

1. Open the VB\Builder or VC\Builder sample
2. Click the **New File** button (on the left side in the toolbar), an empty skin is created.
3. Locate the **Background** tool window and select the **Picture\Add New** item in the

menu, the Open file dialog is opened.
4. Select the picture file (GIF, BMP, JPG, JPEG). You will notice that the visual appearance of the focused object in the skin is changed, actually the picture you have selected is tiled on the object's background.
5. Select the **None** item, in the Background tool window, so the focused object in the skin is not displaying anymore the picture being added.
6. Select the **Root** item in the skin builder window (in the left side you can find the hierarchy of the objects that composes the skin), so the Root item is selected, and so focused.
7. Select the picture file you have added at the step 4, so the Root object is filled with the picture you have chosen.
8. Resize the picture in the Background tool window, until you reach the view you want to have, no black area, or change the CX and CY fields in the Background tool window, so no black area is displayed.
9. Select **Stretch** button in the Background tool window, so the Root object stretches the picture you have selected.
10. Click the **Save a file** button, and select a name for the new skin, click the Save button after you typed the name of the skin file. Add the .ebn extension.
11. Close the builder

You can always open the skin with the builder and change it later, in case you want to change it.

Now, create a new project, and insert the component where you want to use the skin, and add the skin file to the Appearance collection of the object, using blue code, by changing the name of the file or the path where you have selected the skin. Once that you have added the skin file to the Appearance collection, you can change the visual appearance for parts of the controls that supports skinning. **Usually the properties that changes the background color for a part of the control supports skinning as well.**

property Tree.SelBackMode as BackModeEnum

Retrieves or sets a value that indicates whether the selection is transparent or opaque.

Type	Description
BackModeEnum	A BackModeEnum expression that indicates whether the selection is transparent or opaque.

By default, the SelBackMode property is exOpaque. Use the SelBackMode property to specify how the selection is shown in the control. Use the SelBackMode property to specify a specify a semi-transparent color so the selected rows do not lose the colors, pictures, when they are selected. Use the SelBackColor property to specify the visual appearance or the background color for selected items. Use the SelForeColor property to specify the selection foreground color. The SingleSel property specifies whether the control supports single or multiple selection. The control fires the SelectionChanged event when user selects an item. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. The FullRowSelect property specifies whether the full item or a single cell is being selected.

The following screen shot shows the control when no items are selected:

The following screen shot shows the first three items selected, while the SelBackMode property is **exOpaque**:

The following screen shot shows the first three items selected, while the SelBackMode property is **exOpaque**, and FullRowSelect property is 0:

The following screen shot shows the first three items selected, while the SelBackMode property is **exTransparent**:

The following screen shot shows the first three items selected, while the SelBackMode property is **exGrid**:

property Tree.SelectColumn as Boolean

Specifies whether the user selects cells only in SelectColumnIndex column, while FullRowSelect property is False.

Type	Description
Boolean	A boolean expression that specifies whether the user selects cells only in SelectColumnIndex column, while the FullRowSelect property is False

By default, the SelectColumn property is False. The SelectColumn property has effect only if the FullRowSelect is False. The control displays the selected cell in the SelectColumnIndex column. The SelectColumnIndex property specifies the index of selected column. Use the SelectableItem property to specify the user can select an item.

property Tree.SelectColumnIndex as Long

Retrieves or sets a value that indicates the column's index where the user can select an item by clicking.

Type	Description
Long	A long expression that indicates the column's index where the user can select the item.

The property has effect only if the FullRowSelect property is False. Use the SelectedItem property to determine the selected items. Use the SelectColumnInner property to get the index of the inner cell that's selected or focused. Use the SplitCell property to split a cell. Use the SelectableItem property to specify the user can select an item.

property Tree.SelectColumnInner as Long

Retrieves or sets a value that indicates the index of the inner cell that's selected.

Type	Description
Long	A long expression that indicates the index of the inner cell that's focused or selected.

Use the SelectColumnInner property to get the index of the inner cell that's selected or focused. The SelectColumnInner property may be greater than zero, if the control contains inner cells. The SplitCell method splits a cell in two cells. The newly created cell is called inner cell. The FocusItem property indicates the focused item. The SelectColumnIndex property determines the index of the column that's selected when FullRowSelect property is False. Use the SelectableItem property to specify the user can select an item.

property Tree.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 Tree.SelForeColor as Color

Retrieves or sets a value that indicates the selection foreground color.

Type	Description
Color	A color expression that indicates the selection foreground color.

Use the SelForeColor and SelBackColor properties to change the colors used for selected items. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. The control fires the SelectionChanged event when user changes the selection. Use the SelectableItem property to specify the user can select an item.

property Tree.SelLength as Long

Returns or sets the number of characters selected.

Type	Description
Long	A long expression that indicates the number of characters selected.

By default, the SelLenght property is -1 (all text gets selected). Use the SelLenght property to specify the number of characters being selected when the edit operations begins. The SelStart and SelLength properties have effect only if the control is editable. Use the AllowEdit property to allow control edits the data using a text box field. Use the Edit method to programmatically edit a cell using a textbox field. The SelLength property must be set in the code, before starting editing the cell. The control fires the BeforeCellEdit event when the control is about to open the text box editor. The control fires the AfterCellEdit property when the edit ends.

property Tree.SelStart as Long

Returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.

Type	Description
Long	A long expression that indicates the starting point of text selected

By default, the SelStart property is 0 (the text gets selected from the first character). Use the SelStart property to specify the starting point of selected text, when edit operations begins. The SelStart and SelLength properties are valid only if the control is editable. Use the AllowEdit property to allow control edits the data using a text box field. Use the Edit method to programmatically edit a cell using a textbox field. The SelStart property must be set in the code, before starting editing the cell. The control fires the BeforeCellEdit event when the control is about to open the text box editor. The control fires the AfterCellEdit property when the edit ends.

property Tree.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 SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same.

property Tree.ShowImageList as Boolean

Specifies whether the control's image list window is visible or hidden.

Type	Description
Boolean	A boolean expression that specifies whether the control's image list window is visible or hidden.

By default, the ShowImageList property is True. Use the ShowImageList property to hide the control's images list window. The control's images list window is visible only at design time. Use the Images method to associate an images list control to the tree control. Use the RepaceIcon method to add, remove or clear icons in the control's images collection. Use the CellImage, CellImages properties to assign icons to a cell. Use the CellPicture property to assign a picture to a cell. Use the CheckImage or RadioImage property to specify a different look for checkboxes or radio buttons in the cells.

property Tree.ShowLockedItems as Boolean

Retrieves or sets a value that indicates whether the locked items are visible or hidden.

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.

method Tree.ShowToolTip (ToolTip as String, [Title as Variant], [Alignment as Variant], [X as Variant], [Y as Variant])

Shows the specified tooltip at given position.

Type	Description
ToolTip as String	The ToolTip parameter can be any of the following: • NULL(BSTR) or "<null>"(string) to indicate that the tooltip for the object being hovered is not changed • A String expression that indicates the description of the tooltip, that supports built-in HTML format (adds, replaces or changes the object's tooltip)
Title as Variant	The Title parameter can be any of the following: • missing (VT_EMPTY, VT_ERROR type) or "<null>" (string) the title for the object being hovered is not changed. • A String expression that indicates the title of the tooltip (no built-in HTML format) (adds, replaces or changes the object's title)
Alignment as Variant	A long expression that indicates the alignment of the tooltip relative to the position of the cursor. If missing (VT_EMPTY, VT_ERROR) the alignment of the tooltip for the object being hovered is not changed. The Alignment parameter can be one of the following: • 0 - exTopLeft • 1 - exTopRight • 2 - exBottomLeft • 3 - exBottomRight • 0x10 - exCenter • 0x11 - exCenterLeft • 0x12 - exCenterRight • 0x13 - exCenterTop • 0x14 - exCenterBottom By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).

X as Variant	Specifies the horizontal position to display the tooltip as one of the following: • missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored) • -1, indicates the current horizontal position of the cursor (current x-position) • a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position) • a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)
Y as Variant	Specifies the vertical position to display the tooltip as one of the following: • missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored) • -1, indicates the current vertical position of the cursor (current y-position) • a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position) • a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the MouseMove event. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to change the tooltip's font. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color.

For instance:

- ShowToolTip(`<null>`,`<null>`,,`+8`,`+8`), shows the tooltip of the object moved relative

to its default position
- ShowToolTip(`<null>`,`new title`), adds, changes or replaces the title of the object's tooltip
- ShowToolTip(`new content`), adds, changes or replaces the object's tooltip
- ShowToolTip(`new content`,`new title`), shows the tooltip and title at current position
- ShowToolTip(`new content`,`new title`,,`+8`,`+8`), shows the tooltip and title moved relative to the current position
- ShowToolTip(`new content`,``,,128,128), displays the tooltip at a fixed position
- ShowToolTip(``,``), hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

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

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

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

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

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

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

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

property Tree.SingleSel as Boolean

Retrieves or sets a value that indicates whether the control supports single or multiple selection.

Type	Description
Boolean	A boolean expression that indicates whether the control supports single or multiple selection.

Use the SingleSel property to enable multiple selection. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. The control fires the SelectionChanged event when user selects an item. Use the SelForeColor and SelBackColor properties to specify colors for selected items. Use the SelectableItem property to specify the user can select an item. The FullRowSelect property specifies whether the selection spans the entire width of the control. Use the SelectAll method to select all visible items in the tree. Use the UnselectAll method to unselect all items in the control.

property Tree.SingleSort as Boolean

Returns or sets a value that indicates whether the control supports sorting by single or multiple columns.

Type	Description
Boolean	A boolean expression that indicates whether the control supports 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 Tree.SortBarCaption as String

Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns.

Type	Description
String	A String expression that indicates the caption of the control's sort bar.

The SortBarCaption property specifies the caption of the control's sort bar, when it contains no sorted columns. Use the SortBarVisible property to show the control's sort bar. Use the BackColorSortBar, BackColorSortBarCaption and ForeColorSortBar properties to specify colors for the control's sort bar. Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarColumnWidth property to specify the width of the column in the control's sort bar. By default, the SortBarCaption property is "Drag a column header here to sort by that column.". Use the Font property to specify the control's font. Use the ItemBySortPosition property to access the columns in the control's sort bar.

The SortBarCaption property may include built-in HTML tags like follows:

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

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

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

Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

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

loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.

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

gradient-center

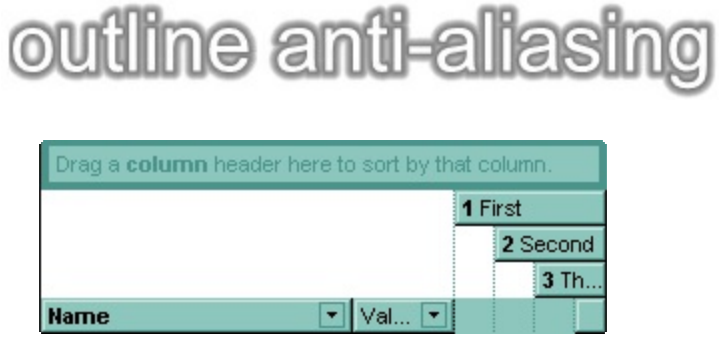
- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text.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

outlined

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

shadow

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



property Tree.SortBarColumnWidth as Long

Specifies the maximum width a column can be in the control's sort bar.

Type	Description
Long	A long expression that indicates the width of the columns in the control's sort bar. If the value is negative, all columns in the sort bar are displayed with the same width (the absolute value represents the width of the columns, in pixels). If the value is positive, it indicates the maximum width, so the width of the columns in the sort bar may differ.

Use the SortBarColumnWidth property to specify the width of the column in the control's sort bar. Use the SortBarVisible property to show the control's sort bar. Use the Width property to specify the width of the column in the control's header bar. Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarCaption property to specify the caption being displayed in the control's sort bar when it contains no columns.

property Tree.SortBarHeight as Long

Retrieves or sets a value that indicates the height of the control's sort bar.

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

Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarVisible property to show the control's sort bar. By default, the SortBarHeight property is 18 pixels. Use the HeaderHeight property to specify the height of the control's header bar. Use the SortBarColumnWidth property to specify the width of the columns being displayed in the control's sort bar. Use the BackColorSortBar, BackColorSortBarCaption and ForeColorSortBar properties to specify colors for the control's sort bar. Use the SortBarCaption property to specify the caption being displayed in the control's sort bar when it contains no columns.

property Tree.SortBarVisible as Boolean

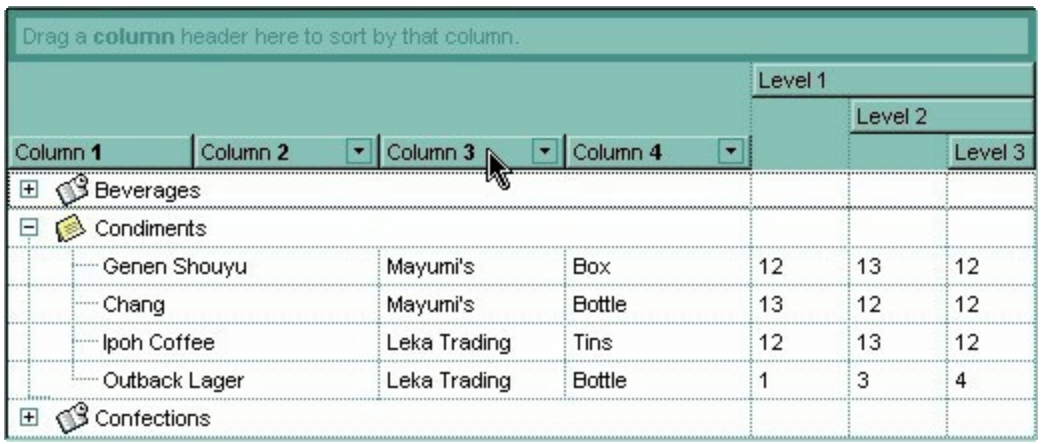
Retrieves or sets a value that indicates whether control's sort bar is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the sort bar is visible or hidden.

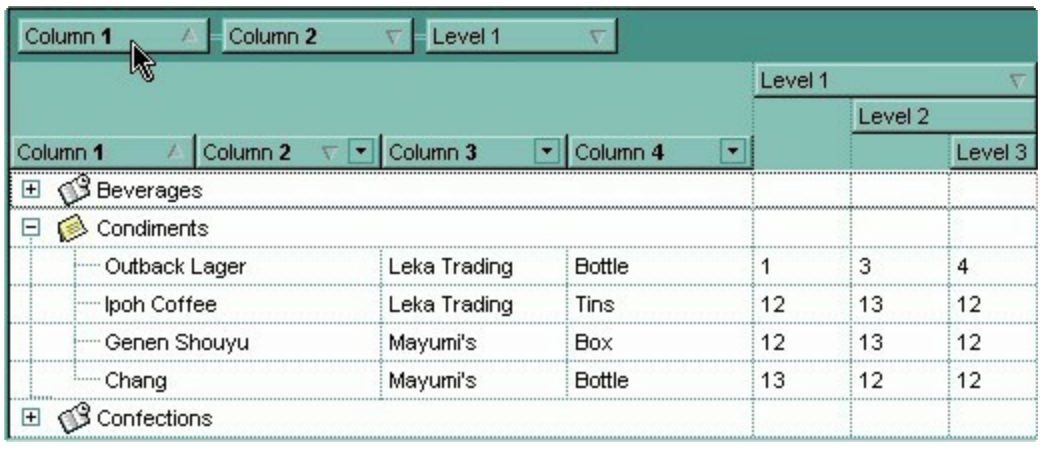
Use the SortBarVisible property to show the control's sort bar. By default, the SortBarVisible property is False. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. Sorting by a single column in the control is a simple matter of clicking on the column head. Sorting by multiple columns, however, is not so obvious. But it's actually quite easy. The user has two options to sort by multiple columns:

- 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 Tree.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 SortOnClick expression that indicates whether the control sorts automatically the data when the user click on the column's header.

Use the SortOnClick property to disable sorting items when the user clicks on the column's header. Use the SortBarVisible property to show the control's sort bar. Use the SingleSort property to allow sorting by single or multiple columns. Use the AllowSort property to avoid sorting a column when user clicks the column. Use the DefaultSortOrder property to specify the column's default sort order, when the user first clicks the column's header.

There are two methods to get the items sorted like follows:

- Using the SortOrder property of the Column object::

  ```
  Tree1.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).

  ```
  Tree1.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 Tree.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 57 = 47,424 (58.02%)
Control: 1 x 19,944 = 19,944 (24.40%)
Column: 13 x 688 = 8,944 (10.94%)
Item: 64 x 72 = 4,608 (5.64%)
Items: 1 x 620 = 620 (0.76%)
Columns: 1 x 172 = 172 (0.21%)
Appearances: 1 x 28 = 28 (0.03%)
Appearance: 0 x 712 = 0 (0.00%)
CComVariant: 0 x 16 = 0 (0.00%)
Cells(Inner): 0 x 57 = 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 97 = 80,704 (58.76%)
Control: 1 x 32,552 = 32,552 (23.70%)
Column: 13 x 1,104 = 14,352 (10.45%)
Item: 64 x 128 = 8,192 (5.96%)
Items: 1 x 1,184 = 1,184 (0.86%)
Columns: 1 x 320 = 320 (0.23%)
Appearances: 1 x 48 = 48 (0.03%)
Appearance: 0 x 1,168 = 0 (0.00%)
CComVariant: 0 x 24 = 0 (0.00%)
Cells(Inner): 0 x 97 = 0 (0.00%)
CSmartVariant: 0 x 9 = 0 (0.00%)
```

property Tree.Template as String

Specifies the control's template.

Type	Description
String	A string expression that indicates the control's template.

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the ToTemplate property to generate the control's content to template format. Use the ExecuteTemplate property to gets the result after 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 Tree.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters)* = *Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

```
with (Control)
   TemplateDef = [Dim var_Column]
   TemplateDef = var_Column
   Template = [var_Column.Def(4) = 255]
endwith
```

This sample allocates a variable var_Column, assigns the value to the variable (the second call of the TemplateDef), and the Template call uses the var_Column variable (as an object), to call its Def property with the parameter 4.

Let's say we need to define the background color for a specified column, so we need to call the Def(exCellBackColor) property of the column, to define the color for all cells in the column.

The following **VB6** sample shows setting the Def property such as:

```
With Control
   .Columns.Add("Column 1").Def(exCellBackColor) = 255
   .Columns.Add "Column 2"
   .Items.AddItem 0
   .Items.AddItem 1
```

```
   .Items.AddItem 2
End With
```

In **dBASE Plus**, calling the Def(4) has no effect, instead using the TemplateDef helps you to use properly the Def property as follows:

```
local Control,var_Column

Control = form.Activex1.nativeObject
// Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
with (Control)
   TemplateDef = [Dim var_Column]
   TemplateDef = var_Column
   Template = [var_Column.Def(4) = 255]
endwith
Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

The equivalent sample for **XBasic in A5**, is as follows:

```
Dim Control as P
Dim var_Column as P

Control = topparent:CONTROL_ACTIVEX1.activex
' Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
Control.TemplateDef = "Dim var_Column"
Control.TemplateDef = var_Column
Control.Template = "var_Column.Def(4) = 255"

Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

The samples just call the Column.Def(4) = Value, using the TemplateDef. The first call of TemplateDef property is "Dim var_Column", which indicates that the next call of the TemplateDef will defines the value of the variable var_Column, in other words, it defines the object var_Column. The last call of the Template property uses the var_Column member to use the x-script and so to set the Def property so a new color is being assigned to the column.

The TemplateDef, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)**....** *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or ` characters. If using the ` character, please

make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

method Tree.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
NewVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

The TemplateDef, TemplatePut, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the*

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by **#** character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by **"** or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(**file**)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property Tree.ToolTipDelay as Long

Specifies the time in ms that passes before the ToolTip appears.

Type	Description
Long	A long expression that specifies the time in ms that passes before the ToolTip appears.

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.

property Tree.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

Type	Description
IFontDisp	A Font object being used to display the tooltip.

Use the ToolTipFont property to assign a font for the control's tooltip. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window. Use the CellToolTip property to specify the cell's tooltip.

property Tree.ToolTipMargin as String

Defines the size of the control's tooltip margins.

Type	Description
String	A string expression that defines the horizontal and vertical margins (separated by comma) of the control's tooltip as one of the following formats: • "value", where value is a positive number, that specifies the horizontal and vertical margins, such as "4" equivalent of "4,4" • "value,", where value is a positive number, that specifies the horizontal margin, such as "4," equivalent of "4,0" • ",value", where value is a positive number, that specifies the vertical margin, such as ",4" equivalent of "0,4" • "horizontal,vertical", where horizontal and vertical are positive numbers, that specifies the horizontal and vertical margins, such as "4,4"

By default, the size of the tooltip margin is "4" (horizontal and vertical). For instance, ToolTipMargin = "8" changes the horizontal and vertical margins are set to 8 pixels. ToolTipMargin = "8,4" changes the horizontal margin to 8 pixels and the vertical margin to 4 pixels. The ToolTipWidth property specifies a value that indicates the width of the tooltip window, in pixels. Use the ShowToolTip method to display a custom tooltip. Use the ToolTipFont property to assign a font for the control's tooltip. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears.

property Tree.ToolTipPopDelay as Long

Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

Type	Description
Long	A long expression that specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

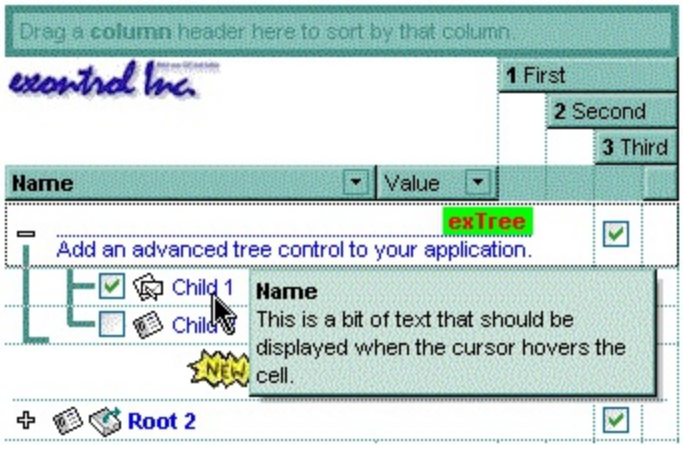
If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to assign a font for the control's tooltip. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.

property Tree.ToolTipWidth as Long

Specifies a value that indicates the width of the tooltip window, in pixels.

Type	Description
Long	A long expression that indicates the width of the tooltip window.

Use the ToolTipWidth property to change the tooltip window width. The height of the tooltip window is automatically computed based on tooltip's description. The ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipFont property to assign a font for the control's tooltip. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color. Use the CellToolTip property to specify the cell's tooltip. Use the ShowToolTip method to display a custom tooltip.



property Tree.ToTemplate ([DefaultTemplate as Variant]) as String

Generates the control's template.

Type	Description
DefaultTemplate as Variant	A String expression that indicates the default format used to define the control's template at runtime, or a string expression that indicates the path to the file being used to define the default template (like c:\temp\teml.bin). If it is missing (by default), the control's uses the default implementation (listed bellow) to define the control's template, at runtime. Each line in the DefaultTemplate parameter, defines a property or an instruction to generate the template.
String	A String expression that indicates the control's template.

Use the ToTemplate property to save the control's content to a template string. The ToTemplate property saves the control's properties based on the default template. Use the ToTemplate property to copy the control's content to another instance. The ToTemplate property can save pictures, icons, binary arrays, objects, collections, and so on based on the DefaultTemplate parameter.

The DefaultTemplate parameter indicates the format of the template being used to generate the control's template at runtime. If the DefaultTemplate parameter is missing, the control's uses its default template listed bellow. The DefaultTemplate parameter defines the list of properties and instructions that generates the control's template. Remove the properties and objects, in the default template, that you don't need in the generated template script. Use the Template property to apply the template to the control. Use the Template property to execute code by passing instructions as a string (template string). The Template script is composed by lines of instructions. Instructions are separated by "\n\r" (newline) characters. The Template format contains a list of instructions that loads data and change properties for the objects in the control. Use the AllowCopyTemplate property to copy the control's content to the clipboard, in template format, using the the Shift + Ctrl + Alt + Insert sequence.

The time to generate the control's template depends on:

- the content of the DefaultTemplate parameter.
- number of columns and items in the control including internal objects such as editors.
- encoding the visual appearance as well as encoding the pictures and icons of the control

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

```
Appearance = 2
AllowEdit = -1
```

In this case the ToTemplate property generates code only for the properties Appearance and AllowEdit, if they were changed to a different value.

If the DefaultTemplate parameters looks like:

```
Appearance
AllowEdit = -1
```

The ToTemplate property always generates code for the Appearance property, and it generates code for the AutoEdit property only if this is changed to a value different than -1. If the DefaultTemplate parameter is missing, the control uses its default template to generate the template format. The default template format looks like follow, and it may differ from a version to another.

```
[0 = BeginUpdate]
VisualAppearance
   [0 = Add(%ID,%CONTENT)]
[1 = Images(%VALUE)]
AllowEdit = 0
Appearance = 2
ASCIILower = "abcdefghijklmnopqrstuvwxyzüéâäŕĺçęëčďîěôöňűůáíóúń"
ASCIIUpper = "ABCDEFGHIJKLMNOPQRSTUVWXYZÜÉÂÄŔĹÇĘËČĎÎĚÔÖŇŰŮÁÍÓÚŃ"
AutoSearch = -1
BackColor = 2147483653
BackColorAlternate = 0
BackColorHeader = 2147483663
BackColorLevelHeader = 2147483663
BackColorLock = 2147483653
BackColorSortBar = 2147483664
BackColorSortBarCaption = 2147483663
Background(3) = 0
Background(2) = 0
Background(8) = 0
Background(12) = 0
Background(11) = 0
Background(14) = 0
```

Background(13) = 0
Background(10) = 0
Background(9) = 0
Background(1) = 0
Background(0) = 0
Background(20) = 0
Background(21) = 0
CheckImage(0) = 0
CheckImage(1) = 0
CheckImage(2) = 0
ColumnAutoResize = -1
ColumnsAllowSizing = 0
ContinueColumnScroll = -1
CountLockedColumns = 0
DefaultItemHeight = 18
Description(0) = "(All)"
Description(11) = " and "
Description(1) = "(Blanks)"
Description(19) = "(Checked)"
Description(12) = "Date:"
Description(17) = "January February March April May June July August September October November December"
Description(15) = "Date"
Description(13) = "to"
Description(16) = "Today"
Description(14) = "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**)."
Description(18) = "S M T W T F S"
Description(3) = "Filter For:"
Description(8) = "A pattern filter may contain the wild card characters '?' for any single character, '*' for zero or more occurrences of any character, '#' for any digit character, '|' determines the options in the pattern. For instance: '1*|2*' specifies all items that start with '1' or '2'. 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."
Description(4) = "Filter"

```
Description(9) = "IsBlank"
Description(21) = "IsChecked"
Description(10) = "not IsBlank"
Description(22) = "not IsChecked"
Description(2) = "(NonBlanks)"
Description(5) = "Pattern/Numeric Filter"
Description(7) = "You can select multiple filter items as many as you like by keeping the CTRL key pressed. Start typing characters if you like to enter a filter as a pattern that may include wild card characters like *,? or #. Press ENTER key to filter the items using the typed pattern. 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."
Description(6) = "You can select multiple filter items as many as you like by keeping the CTRL key pressed. "
Description(20) = "(Unchecked)"
DetectAddNew = 0
DrawGridLines = 0
Enabled = -1
ExpandOnDblClick = -1
ExpandOnKeys = -1
ExpandOnSearch = 0
FilterBarBackColor = 2147483663
FilterBarCaption = ""
FilterBarDropDownHeight = 0.5
FilterBarFont
   Bold = -1
   Charset = 0
   Italic = 0
   Name = "Arial"
   Size = 8.25
   Strikethrough = 0
   Underline = 0
   Weight = 700
FilterBarForeColor = 2147483656
FilterBarHeight = -1
FilterInclude = 0
Font
   Bold = 0
```

```
   Charset = 0
   Italic = 0
   Name = "Arial"
   Size = 8.25
   Strikethrough = 0
   Underline = 0
   Weight = 400
ForeColor = 2147483656
ForeColorHeader = 2147483656
ForeColorLock = 2147483656
ForeColorSortBar = 2147483664
FullRowSelect = -1
GridLineColor = 8949832
HasButtons = -1
HasButtonsCustom(0) = 0
HasButtonsCustom(-1) = 0
HasLines = -1
HeaderAppearance = 3
HeaderHeight = 18
HeaderVisible = -1
HideSelection = 0
HyperLinkColor = 16737585
Indent = 22
ItemsAllowSizing = 0
LinesAtRoot = 0
MarkSearchColumn = -1
OLEDropMode = 0
PictureDisplay = 48
[255 = Picture = LoadPicture("%VALUE")]
PictureDisplayLevelHeader = 48
[256 = PictureLevelHeader = LoadPicture("%VALUE")]
RadioImage(0) = 0
RadioImage(-1) = 0
RClickSelect = 0
ScrollBars = 3
ScrollBySingleLine = 0
SearchColumnIndex = 0
```

SelBackColor = 2147483661
SelBackMode = 0
SelectColumn = 0
SelectColumnIndex = 0
SelectColumnInner = 0
SelForeColor = 2147483662
SelLength = -1
SelStart = 0
ShowFocusRect = -1
ShowImageList = 0
ShowLockedItems = -1
SingleSel = -1
SingleSort = -1
SortBarCaption = "Drag a **column** header here to sort by that column."
SortBarColumnWidth = -96
SortBarHeight = 18
SortBarVisible = 0
SortOnClick = -1
ToolTipDelay = 500
ToolTipPopDelay = 5000
ToolTipWidth = 196
TreeColumnIndex = 0
UseTabKey = -1
Columns
  _NewEnum = "%Caption"
    Alignment = 0
    AllowDragging = -1
    AllowSizing = -1
    AllowSort = -1
    AutoSearch = 0
    [AutoWidth = 0]
    Caption = ""
    Data
    Def(4)
    Def(3) = 0
    Def(17) = 0
    Def(5)

Def(2) = 0
Def(0) = 0
Def(1) = 0
Def(16) = -1
DefaultSortOrder = 0
DisplayFilterButton = 0
DisplayFilterDate = 0
DisplayFilterPattern = -1
DisplaySortIcon = -1
Enabled = -1
Filter = ""
FilterBarDropDownWidth = 1
FilterList = 0
FilterType = 0
FireFormatColumn = 0
HeaderAlignment = 0
HeaderBold = 0
HeaderImage = 0
HeaderImageAlignment = 0
HeaderItalic = 0
HeaderStrikeOut = 0
HeaderUnderline = 0
HTMLCaption = ""
[Index = 0]
Key = ""
LevelKey
MaxWidthAutoResize = -1
MinWidthAutoResize = 0
PartialCheck = 0
Position
SortOrder = 0
SortPosition = -1
SortType = 0
ToolTip = "..."
Visible = -1
Width
WidthAutoResize = 0

```
Items
   PathSeparator = "\"
   LockedItemCount(0) = 0
   LockedItemCount(2) = 0
   [1 = %H = LockedItem(%A,%I)]
      [10 = CellCaption(%H,%C) = %VALUE]
      [11 = CellImage(%H,%C) = %VALUE]
      [12 = CellSingleLine(%H,%C) = %VALUE]
      [13 = CellCaptionFormat(%H,%C) = %VALUE]
      [14 = CellFormatLevel(%H,%C) = %VALUE]
      [15 = CellHasCheckBox(%H,%C) = %VALUE]
      [16 = CellHasRadioButton(%H,%C) = %VALUE]
      [17 = CellState(%H,%C) = %VALUE]
      [18 = CellToolTip(%H,%C) = %VALUE]
      [19 = CellHasButton(%H,%C) = %VALUE]
      [20 = CellButtonAutoWidth(%H,%C) = %VALUE]
      [21 = CellEnabled(%H,%C) = %VALUE]
      [23 = CellHAlignment(%H,%C) = %VALUE]
      [24 = CellVAlignment(%H,%C) = %VALUE]
      [25 = CellMerge(%H,%C) = %VALUE]
      [26 = CellBold(%H,%C) = %VALUE]
      [27 = CellItalic(%H,%C) = %VALUE]
      [28 = CellUnderline(%H,%C) = %VALUE]
      [29 = CellStrikeOut(%H,%C) = %VALUE]
      [30 = CellForeColor(%H,%C) = %VALUE]
      [31 = CellBackColor(%H,%C) = %VALUE]
      [32 = CellPicture(%H,%C) = %VALUE]
      [33 = CellPictureWidth(%H,%C) = %VALUE]
      [34 = CellPictureHeight(%H,%C) = %VALUE]
      [1000 = ExpandItem(%H) = %VALUE]
      [1001 = SelectItem(%H) = %VALUE]
      [1002 = ItemHeight(%H) = %VALUE]
      [1003 = ItemDivider(%H) = %VALUE]
      [1004 = ItemDividerLine(%H) = %VALUE]
      [1005 = ItemDividerLineAlignment(%H) = %VALUE]
      [1006 = ItemHasChildren(%H) = %VALUE]
      [1007 = ItemBold(%H) = %VALUE]
```

[1008 = ItemItalic(%H) = %VALUE]
[1009 = ItemUnderline(%H) = %VALUE]
[1010 = ItemStrikeOut(%H) = %VALUE]
[1011 = ItemForeColor(%H) = %VALUE]
[1012 = ItemBackColor(%H) = %VALUE]
[0 = %H = %ADD(%VALUE)]
[10 = CellCaption(%H,%C) = %VALUE]
[11 = CellImage(%H,%C) = %VALUE]
[12 = CellSingleLine(%H,%C) = %VALUE]
[13 = CellCaptionFormat(%H,%C) = %VALUE]
[14 = CellFormatLevel(%H,%C) = %VALUE]
[15 = CellHasCheckBox(%H,%C) = %VALUE]
[16 = CellHasRadioButton(%H,%C) = %VALUE]
[17 = CellState(%H,%C) = %VALUE]
[18 = CellToolTip(%H,%C) = %VALUE]
[19 = CellHasButton(%H,%C) = %VALUE]
[20 = CellButtonAutoWidth(%H,%C) = %VALUE]
[21 = CellEnabled(%H,%C) = %VALUE]
[23 = CellHAlignment(%H,%C) = %VALUE]
[24 = CellVAlignment(%H,%C) = %VALUE]
[25 = CellMerge(%H,%C) = %VALUE]
[26 = CellBold(%H,%C) = %VALUE]
[27 = CellItalic(%H,%C) = %VALUE]
[28 = CellUnderline(%H,%C) = %VALUE]
[29 = CellStrikeOut(%H,%C) = %VALUE]
[30 = CellForeColor(%H,%C) = %VALUE]
[31 = CellBackColor(%H,%C) = %VALUE]
[32 = CellPicture(%H,%C) = %VALUE]
[33 = CellPictureWidth(%H,%C) = %VALUE]
[34 = CellPictureHeight(%H,%C) = %VALUE]
[1000 = ExpandItem(%H) = %VALUE]
[1001 = SelectItem(%H) = %VALUE]
[1002 = ItemHeight(%H) = %VALUE]
[1003 = ItemDivider(%H) = %VALUE]
[1004 = ItemDividerLine(%H) = %VALUE]
[1005 = ItemDividerLineAlignment(%H) = %VALUE]
[1006 = ItemHasChildren(%H) = %VALUE]

```
      [1007 = ItemBold(%H) = %VALUE]
      [1008 = ItemItalic(%H) = %VALUE]
      [1009 = ItemUnderline(%H) = %VALUE]
      [1010 = ItemStrikeOut(%H) = %VALUE]
      [1011 = ItemForeColor(%H) = %VALUE]
      [1012 = ItemBackColor(%H) = %VALUE]
      [1013 = SelectableItem(%H) = %VALUE]
[2 = ApplyFilter]
ScrollPos(0) = 0
ScrollPos(-1) = 0
[0 = EndUpdate]
```

For instance, let's say that we need to save the layout (size and position) of the columns (4 columns) in the control. In this case, we need to define a new DefaultTemplate parameter that includes only the Columns section as follows:

```
Columns
   Item(0)
      Position
      Width = 64
   Item(1)
      Position
      Width = 64
   Item(2)
      Position
      Width = 64
   Item(3)
      Position
      Width = 64
```

The indentation in the template is very important, so please make sure that you respect the indentation of the inside objects and properties. If an item in the template is indented it is related to the parent item/object.

property Tree.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 the column where the control's hierarchy is displayed.

Use the TreeColumnIndex property to change the column's index where the hierarchy lines are painted. Use HasLines and LinesAtRoot properties to show the hierarchy lines. Use the HasButtons property to define the +/- signs appearance. If the TreeColumnIndex property is -1, the control doesn't paint the hierarchy. Use the Indent property to define the amount, in pixels, that child items are indented relative to their parent items.

property Tree.UseTabKey as Boolean

Specifies whether the TAB key is used to change the searching column.

Type	Description
Boolean	A boolean expression that specifies whether the TAB key is used to change the incremental searching column.

By default, the UseTabKey property is True. The UseTabKey property specifies whether the control uses the TAB key to change the searching column. If the UseTabKey property is False, the TAB key is used to navigate through the form's controls.

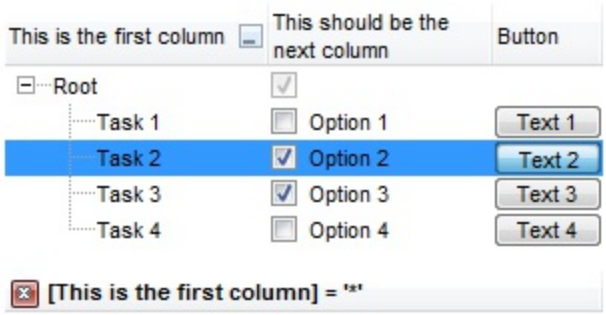
property Tree.UseVisualTheme as UIVisualThemeEnum

Specifies whether the control uses the current visual theme to display certain UI parts.

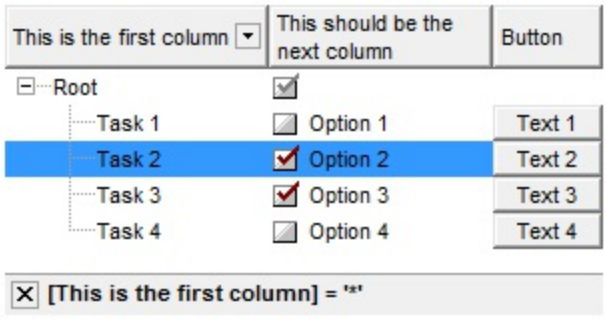
Type	Description
UIVisualThemeEnum	An UIVisualThemeEnum expression that specifies which UI parts of the control are shown using the current visual theme.

By default, the UseVisualTheme property is exDefaultVisualTheme, which means that all known UI parts are shown as in the current theme. The UseVisualTheme property may specify the UI parts that you need to enable or disable the current visual theme. The UI Parts are like header, filterbar, check-boxes, buttons and so on. The UseVisualTheme property has effect only a current theme is selected for your desktop. The UseVisualTheme property. Use the Appearance property of the control to provide your own visual appearance using the EBN files.

The following screen shot shows the control while the UseVisualTheme property is exDefaultVisualTheme:



since the second screen shot shows the same data as the UseVisualTheme property is exNoVisualTheme:



property Tree.Version as String

Retrieves the control's version.

Type	Description
String	A string expression that indicates the control's version.

The version property specifies the control's version.

property Tree.VisualAppearance as Appearance

Retrieves the control's appearance.

Type	Description
Appearance	An Appearance object that holds a collection of skins.

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (*.ebn) assigned to a part of the control. By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part.

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

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

property Tree.VisualDesign as String

Invokes the control's VisualAppearance designer.

Type	Description
String	A String expression that encodes the control's Visual Appearance.

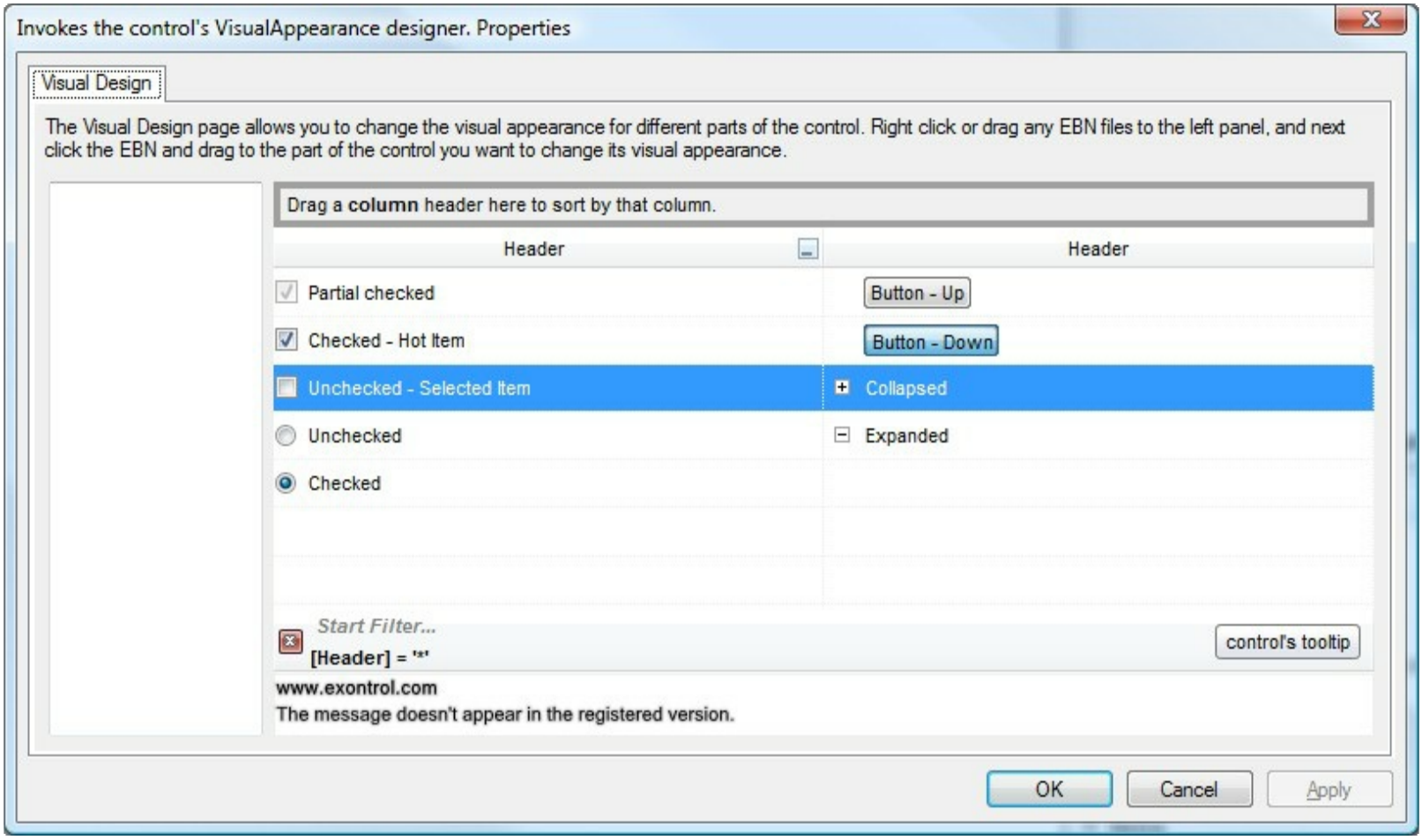
By default, the VisualDesign property is "". The VisualDesign property helps you to define fast and easy the control's visual appearance using the XP-Theme elements or EBN objects. The VisualDesign property can be accessed on design mode, and it can be used to design the visual appearance of different parts of the control by drag and drop XP or EBN elements. The VisualAppearance designer returns an encoded string that can be used to define different looks, just by calling the VisualDesign = encoded_string. If you require removing the current visual appearance, you can call the VisualDesign on "" (empty string). The VisualDesign property encodes EBN or XP-Theme nodes, using the Add method of the Appearance collection being accessed through the VisualAppearance property.

- For the /COM version, click the control in Design mode, select the Properties, and choose the "Visual Design" page.
- For the /NET version, select the VisualDesign property in the Properties browser, and then click ... so the "Visual Design" page is displayed.
- The /WPF version does not provide a VisualAppearance designer, instead you can use the values being generated by the /COM or /NET to apply the same visual appearance.

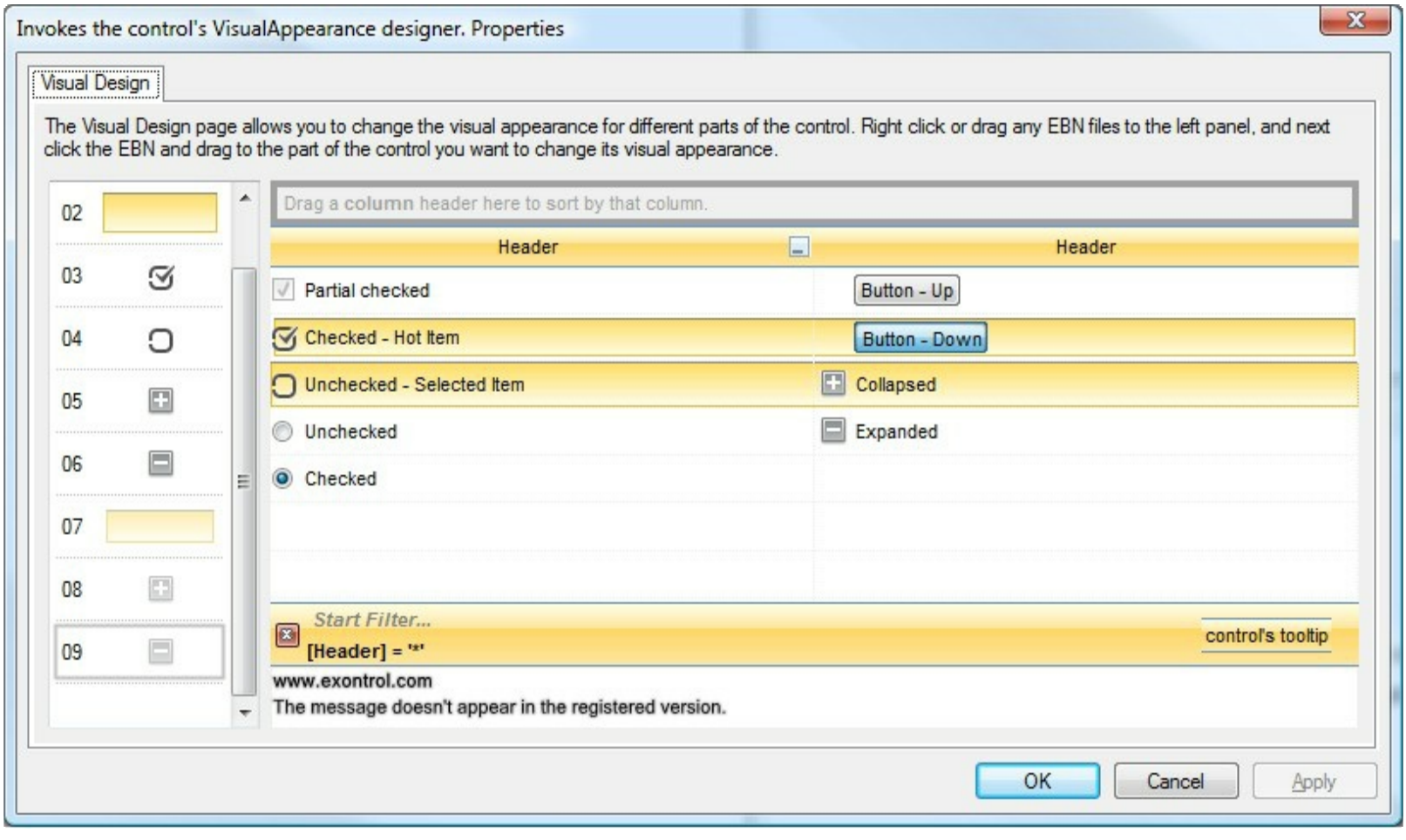
- Click here to watch a movie on how you define the control's visual appearance using the XP-Theme
- Click here to watch a movie on how you define the control's visual appearance using the EBN files.

The left panel, should be user to add your EBN or XP-Theme elements. Once you add them drag and drop the EBN or XP-Theme element from the left side to the part which visual appearance you want to change.

The following picture shows the control's VisualDesign form (empty):



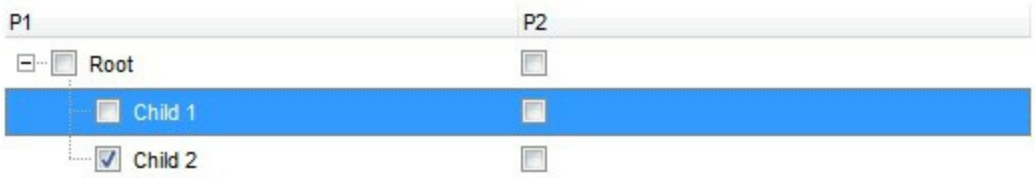
The following picture shows the control's VisualDesign form after applying some EBN objects:



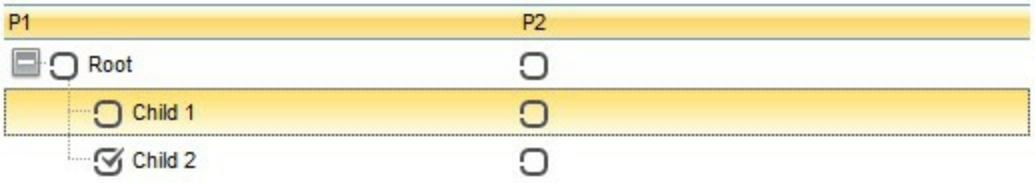
This layout generates the following code:

```
With Extree1
   .VisualDesign =
"gBFLBWIgBAEHhEJAEGg7oB0HBSQAwABsIfj/jEJAcKhYEjgCAscA8ThQBA8cAgIjgDh8KBAPj
 & _

"RuF6FxmAkchiheZg5gYZIW0yMhZhqD55jIboamcCY2HGG5nCmVh0h2ZYUAyCQ4Xqbh9h8
 & _

"o5B8MwE4HsD4/g/ijHQHoLwrxUjrH0H4Z4rR2h7A8N8UggRNBnGCP8eA/A/gXGSPMfg3w
 & _

"DCDgJQFICxhDQGYBofYQYFCwD4J+XYQwIBECiCwJIExhhnCIDoNAnhzj8CyBcIosQ+BlAwM
 & _

"J8YQIwaBMCaCMd6hRnBpE+HoIwIQ9hdEKM8VYawoCcC8BUSYtxqBuDuFsOwTgLgLhZhAh
 & _

"zhGhtoEB+AsArhnhLhehUB5BfA4BfARBPgWB9h3hhBZB/AvA+BzhkhLhCh7hPg8g1BfhzAKE
 & _

"hQH1hSgAgcAmghgIg2AugLBigiBqAnAzBiVdglA1ANAjBEgbAmAJMwA+gLgjgyBWA4A0E
 & _

"IAUgCA0AMhjA0ggWUgjh+GhBihI1yAKhiByBqAkV1gCAKAiV3141516g+Jmhj19V+V/AI2A

End With
```

If running the empty control we get the following picture:



If running the control using the code being generated by the VisualAppearance designer we get:


P1
P2
Root
Child 1
Child 2

ExTree events

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {3C5FC763-72BA-4B97-9985-81862E9251F2}. The object's program identifier is: "Exontrol.Tree". The /COM object module is: "ExTree.dll"

The ExTree component supports the following events:

Name	Description
AddColumn	Fired after a new column has been added.
AddItem	Occurs after a new Item has been inserted to Items collection.
AfterCellEdit	Occurs after data in the current cell is edited.
AfterExpandItem	Fired after an item is expanded (collapsed).
AllowAutoDrag	Occurs when the user drags the item between InsertA and InsertB as child of NewParent.
AnchorClick	Occurs when an anchor element is clicked.
BeforeCellEdit	Occurs just before the user enters edit mode by clicking in a cell.
BeforeExpandItem	Fired before an item is about to be expanded (collapsed).
CancelCellEdit	Occurs if the edit operation is canceled.
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.
Click	Occurs when the user presses and then releases the left mouse button over the tree control.
ColumnClick	Fired after the user clicks on column's header.
DblClick	Occurs when the user dblclk the left mouse button over an object.
Event	Notifies the application once the control fires an event.
FilterChange	Notifies your application that the filter is changed.
FilterChanging	Notifies your application that the filter is about to change.
FormatColumn	Fired when a cell requires to format its caption.
HyperLinkClick	Occurs when the user clicks on a hyperlink cell.
	Fired when an ActiveX control hosted by an item has fired

ItemOleEvent	an event.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
LayoutChanged	Occurs when column's position or column's size is changed.
MouseDown	Occurs when the user presses a mouse button.
MouseMove	Occurs when the user moves the mouse.
MouseUp	Occurs when the user releases a mouse button.
OffsetChanged	Occurs when the scroll position has been changed.
OLECompleteDrag	Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled
OLEDragDrop	Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.
OLEDragOver	Occurs when one component is dragged over another.
OLEGiveFeedback	Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.
OLESetData	Occurs on a drag source when a drop target calls the GetData method and there is no data in a specified format in the OLE drag-and-drop DataObject.
OLEStartDrag	Occurs when the OLEDrag method is called.
OversizeChanged	Occurs when the right range of the scroll has been changed.
RClick	Fired when right mouse button is clicked
RemoveColumn	Fired before deleting a Column.
RemoveItem	Occurs before deleting an Item.
ScrollButtonClick	Occurs when the user clicks a button in the scrollbar.
SelectionChanged	Fired after a new item 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's added to the Columns collection.

The AddColumn event is fired after a new column has been inserted to Columns collection. Use the AddColumn event to associate extra data to a new column. Use the Add method to add new columns to Columns collection. Use the ColumnAutoSize property to fit all visible columns in the control's client area.

Syntax for AddColumn event, **/NET** version, on:

C#
```
private void AddColumn(object sender,exontrol.EXTREELib.Column Column)
{
}
```

VB
```
Private Sub AddColumn(ByVal sender As System.Object,ByVal Column As
exontrol.EXTREELib.Column) Handles AddColumn
End Sub
```

Syntax for AddColumn event, **/COM** version, on:

C#
```
private void AddColumn(object sender,
AxEXTREELib._ITreeEvents_AddColumnEvent e)
{
}
```

C++
```
void OnAddColumn(LPDISPATCH Column)
{
}
```

C++ Builder
```
void __fastcall AddColumn(TObject *Sender,Extreelib_tlb::IColumn *Column)
{
}
```

Delphi
```
procedure AddColumn(ASender: TObject; Column : IColumn);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AddColumn(sender: System.Object; e:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_AddColumnEvent) Handles AddColumn
End Sub
```

VB6

```
Private Sub AddColumn(ByVal Column As EXTREELibCtl.IColumn)
End Sub
```

VBA

```
Private Sub AddColumn(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnAddColumn(oTree,Column)
RETURN
```

Syntax for AddColumn event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AddColumn(Column)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AddColumn(Column)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAddColumn Variant llColumn
    Forward Send OnComAddColumn llColumn
End_Procedure
```

Visual Objects

```
METHOD OCX_AddColumn(Column) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddColumn(COM _Column)
{
}
```

XBasic

```
function AddColumn as v (Column as OLE::Exontrol.Tree.1::IColumn)
end function
```

dBASE

```
function nativeObject_AddColumn(Column)
return
```

The following VB sample shows how to set the width for all columns:

```
Private Sub Tree1_AddColumn(ByVal Column As EXTREELibCtl.IColumn)
    Column.Width = 128
End Sub
```

The following VB.NET sample changes the column's width when a new column is added:

```
Private Sub AxTree1_AddColumn(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_AddColumnEvent) Handles AxTree1.AddColumn
    e.column.Width = 128
End Sub
```

The following C# sample changes the column's width when a new column is added:

```
private void axTree1_AddColumn(object sender,
AxEXTREELib._ITreeEvents_AddColumnEvent e)
{
    e.column.Width = 128;
}
```

The following C++ sample changes the column's width when a new column is added:

```
#include "Column.h"
```

```
#include "Columns.h"
void OnAddColumnTree1(LPDISPATCH Column)
{
    CColumn column( Column );column.m_bAutoRelease = FALSE;
    column.SetWidth( 128 );
}
```

The following VFP sample changes the column's width when a new column is added:

```
*** ActiveX Control Event ***
LPARAMETERS column

with column
    .Width = 128
endwith
```

event AddItem (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 item that's inserted to the Items collection.

The AddItem event notifies your application that a new items is inserted. Use the AddItem and InsertItem methods to insert new items to Items collection. Use the InsertControlItem method to add a new item that hosts an ActiveX control. Use the Add method to add new columns to Columns Collection. Use the Def property to specify a common value for all cells in the same column.

Syntax for AddItem event, **/NET** version, on:

C#
```
private void AddItem(object sender,int Item)
{
}
```

VB
```
Private Sub AddItem(ByVal sender As System.Object,ByVal Item As Integer)
Handles AddItem
End Sub
```

Syntax for AddItem event, **/COM** version, on:

C#
```
private void AddItem(object sender, AxEXTREELib._ITreeEvents_AddItemEvent e)
{
}
```

C++
```
void OnAddItem(long Item)
{
}
```

C++ Builder
```
void __fastcall AddItem(TObject *Sender,Extreelib_tlb::HITEM Item)
{
}
```

Delphi
```
procedure AddItem(ASender: TObject; Item : HITEM);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure AddItem(sender: System.Object; e:
AxEXTREELib._ITreeEvents_AddItemEvent);
begin
end;
```

Powe…

```
begin event AddItem(long Item)
end event AddItem
```

VB.NET

```
Private Sub AddItem(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_AddItemEvent) Handles AddItem
End Sub
```

VB6

```
Private Sub AddItem(ByVal Item As EXTREELibCtl.HITEM)
End Sub
```

VBA

```
Private Sub AddItem(ByVal Item As Long)
End Sub
```

VFP

```
LPARAMETERS Item

```

Xbas…

```
PROCEDURE OnAddItem(oTree,Item)
RETURN
```

Syntax for AddItem event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AddItem(Item)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AddItem(Item)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAddItem HITEM llItem
    Forward Send OnComAddItem llItem
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_AddItem(Item) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddItem(int _Item)
{
}
```

XBasic

```
function AddItem as v (Item as OLE::Exontrol.Tree.1::HITEM)
end function
```

dBASE

```
function nativeObject_AddItem(Item)
return
```

The following VB sample shows how to change the item's foreground color:

```
Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
    Tree1.Items.ItemForeColor(Item) = vbBlue
End Sub
```

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

```
Tree1.Columns(0).Def(exCellBackColor) = RGB(240, 240, 240)
```

The following C++ sample changes the item's foreground color when a new items is inserted:

```
#include "Items.h"
void OnAddItemTree1(long Item)
{
    if ( ::IsWindow( m_tree.m_hWnd ) )
    {
        CItems items = m_tree.GetItems();
        items.SetItemForeColor( Item, RGB(0,0,255) );
    }
}
```

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

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

The following VB.NET sample changes the item's foreground color when a new items is inserted:

```
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

Private Sub AxTree1_AddItem(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_AddItemEvent) Handles AxTree1.AddItem
    AxTree1.Items.ItemForeColor(e.item) = ToUInt32(Color.Blue)
End Sub
```

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

```
With AxTree1.Columns(0)
    .Def(EXTREELib.DefColumnEnum.exCellBackColor) = ToUInt32(Color.WhiteSmoke)
End With
```

The following C# sample changes the item's foreground color when a new items is inserted:

```
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);
}

private void axTree1_AddItem(object sender, AxEXTREELib._ITreeEvents_AddItemEvent e)
{
```

```
    axTree1.Items.set_ItemForeColor( e.item, ToUInt32(Color.Blue) );
}
```

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

```
axTree1.Columns[0].set_Def(EXTREELib.DefColumnEnum.exCellBackColor,
ToUInt32(Color.WhiteSmoke));
```

The following VFP sample changes the item's foreground color when a new items is inserted:

```
*** ActiveX Control Event ***
LPARAMETERS item

with thisform.Tree1.Items
    .DefaultItem = item
    .ItemForeColor( 0 ) = RGB(0,0,255 )
endwith
```

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

```
with thisform.Tree1.Columns(0)
    .Def( 4 ) = RGB(240,240,240)
endwith
```

For instance, the following VB sample loads an ADO recordset.

```
Dim rs As Object

Private Sub Form_Load()

    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

    Tree1.BeginUpdate
    ' Add the columns
    With Tree1.Columns
    For Each f In rs.Fields
```

```
      .Add f.Name
   Next
   End With

   ' Add the items
   With Tree1.Items
   rs.MoveFirst
   While Not rs.EOF
      .InsertItem , rs.Bookmark
      rs.MoveNext
   Wend
End With

   Tree1.EndUpdate
End Sub

Private Sub Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
   Dim i As Integer
   Dim n As Integer
   n = Tree1.Columns.Count
   With Tree1.Items
   For i = 0 To n - 1
      .CellCaption(Item, i) = rs(i).Value
   Next
   End With
End Sub
```

The following VB sample use the PutItems method to load items to the control:

```
Dim rs As Object

Private Sub Form_Load()

   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
```

```
    Tree1.BeginUpdate
    ' Add the columns
    With Tree1.Columns
    For Each f In rs.Fields
        .Add f.Name
    Next
    End With

    Tree1.PutItems rs.getRows()

    Tree1.EndUpdate
End Sub
```

event AfterCellEdit (Item as HITEM, ColIndex as Long, NewCaption as String)

Occurs after data in the current cell is edited.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being changed.
ColIndex as Long	A long expression that specifies the index of the column where the change occurs, or a handle to a cell being edited if the Item parameter is 0.
NewCaption as String	A string expression that indicates the newly cell's caption.

The AfterCellEdit and BeforeCellEdit events are fired only if the AllowEdit property of the tree control is True. Use the Edit method to programmatically edits a cell. The user must handle the AfterCellEdit event else the cell's caption remains unchanged. Use the AfterCellEdit event to change the cell's caption after user edits a cell. The AfterCellEdit event is not fired if the user canceled the edit operation using BeforeCellEdit event. The CancelCellEdit event occurs if the user cancels the edit operation. The CancelCellEdit event is fired when the user presses ESC key while editing or when the user clicks outside the edit field.

Syntax for AfterCellEdit event, **/NET** version, on:

C#
```
private void AfterCellEdit(object sender,int Item,int ColIndex,string NewCaption)
{
}
```

VB
```
Private Sub AfterCellEdit(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer,ByVal NewCaption As String) Handles
AfterCellEdit
End Sub
```

Syntax for AfterCellEdit event, **/COM** version, on:

C#
```
private void AfterCellEdit(object sender,
AxEXTREELib._ITreeEvents_AfterCellEditEvent e)
{
}
```

C++

```
void OnAfterCellEdit(long Item,long ColIndex,LPCTSTR NewCaption)
{
}
```

C++ Builder

```
void __fastcall AfterCellEdit(TObject *Sender,Extreelib_tlb::HITEM Item,long
ColIndex,BSTR NewCaption)
{
}
```

Delphi

```
procedure AfterCellEdit(ASender: TObject; Item : HITEM;ColIndex :
Integer;NewCaption : WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterCellEdit(sender: System.Object; e:
AxEXTREELib._ITreeEvents_AfterCellEditEvent);
begin
end;
```

Powe…

```
begin event AfterCellEdit(long Item,long ColIndex,string NewCaption)
end event AfterCellEdit
```

VB.NET

```
Private Sub AfterCellEdit(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_AfterCellEditEvent) Handles AfterCellEdit
End Sub
```

VB6

```
Private Sub AfterCellEdit(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As
Long,ByVal NewCaption As String)
End Sub
```

VBA

```
Private Sub AfterCellEdit(ByVal Item As Long,ByVal ColIndex As Long,ByVal
NewCaption As String)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,NewCaption
```

Xbas…
```
PROCEDURE OnAfterCellEdit(oTree,Item,ColIndex,NewCaption)
RETURN
```

Syntax for AfterCellEdit event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="AfterCellEdit(Item,ColIndex,NewCaption)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function AfterCellEdit(Item,ColIndex,NewCaption)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComAfterCellEdit HITEM llItem Integer llColIndex String
llNewCaption
    Forward Send OnComAfterCellEdit llItem llColIndex llNewCaption
End_Procedure
```

Visual Objects
```
METHOD OCX_AfterCellEdit(Item,ColIndex,NewCaption) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_AfterCellEdit(int _Item,int _ColIndex,str _NewCaption)
{
}
```

XBasic
```
function AfterCellEdit as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as
N,NewCaption as C)
end function
```

dBASE
```
function nativeObject_AfterCellEdit(Item,ColIndex,NewCaption)
return
```

The following VB sample changes the cell's caption as soon as the edit operation ends.

```
Private Sub Tree1_AfterCellEdit(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long, ByVal NewCaption As String)
```

```
    Tree1.Items.CellCaption(Item, ColIndex) = NewCaption
End Sub
```

Use the BeforeCellEdit is you need to cancel editing cells. The following VB sample cancels editing of any cell that' shoted by the first column:

```
Private Sub Tree1_BeforeCellEdit(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long, Value As Variant, Cancel As Variant)
    Cancel = ColIndex = 0
End Sub
```

The following VB.NET sample changes the cell's caption as soon as the edit operation ends.

```
Private Sub AxTree1_AfterCellEdit(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_AfterCellEditEvent) Handles AxTree1.AfterCellEdit
    AxTree1.Items.CellCaption(e.item, e.colIndex) = e.newCaption
End Sub
```

The following C# sample changes the cell's caption as soon as the edit operation ends.

```
private void axTree1_AfterCellEdit(object sender,
AxEXTREELib._ITreeEvents_AfterCellEditEvent e)
{
    axTree1.Items.set_CellCaption( e.item, e.colIndex, e.newCaption );
}
```

The following C++ sample changes the cell's caption as soon as the edit operation ends.

```
void OnAfterCellEditTree1(long Item, long ColIndex, LPCTSTR NewCaption)
{
    m_tree.GetItems().SetCellCaption( COleVariant( Item ), COleVariant( ColIndex ),
COleVariant( NewCaption ) );
}
```

The following VFP sample changes the cell's caption as soon as the edit operation ends.

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, newcaption
```

```
with thisform.Tree1.Items
    .DefaultItem = item
    .CellCaption( 0, colindex ) = newcaption
endwith
```

event AfterExpandItem (Item as HITEM)

Fired after an item is expanded (collapsed).

Type	Description
Item as HITEM	A long expression that indicates the item's handle that indicates the item expanded or collapsed.

The AfterExapndItem 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 expand or collapsed. The ItemChild property retrieves the first child item. Use the BeforeExpandItem event to cancel expanding or collapsing items.

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,
AxEXTREELib._ITreeEvents_AfterExpandItemEvent e)
{
}
```

C++

```
void OnAfterExpandItem(long Item)
{
}
```

C++ Builder

```
void __fastcall AfterExpandItem(TObject *Sender,Extreelib_tlb::HITEM Item)
{
}
```

Delphi

```
procedure AfterExpandItem(ASender: TObject; Item : HITEM);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterExpandItem(sender: System.Object; e:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_AfterExpandItemEvent) Handles AfterExpandItem
End Sub
```

VB6

```
Private Sub AfterExpandItem(ByVal Item As EXTREELibCtl.HITEM)
End Sub
```

VBA

```
Private Sub AfterExpandItem(ByVal Item As Long)
End Sub
```

VFP

```
LPARAMETERS Item

```

Xbas…

```
PROCEDURE OnAfterExpandItem(oTree,Item)
RETURN
```

Syntax for AfterExpandItem event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AfterExpandItem(Item)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AfterExpandItem(Item)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAfterExpandItem HITEM llItem
    Forward Send OnComAfterExpandItem llItem
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.Tree.1::HITEM)
end function
```

dBASE

```
function nativeObject_AfterExpandItem(Item)
return
```

The following VB sample prints the item's state when it is expanded or collapsed:

```
Private Sub Tree1_AfterExpandItem(ByVal Item As EXTREELibCtl.HITEM)
    Debug.Print "The " & Item & " item was " & Iif(Tree1.Items.ExpandItem(Item),
"expanded", "collapsed")
End Sub
```

The following C# sample prints the item's state when it is expanded or collapsed:

```
private void axTree1_AfterExpandItem(object sender,
AxEXTREELib._ITreeEvents_AfterExpandItemEvent e)
{
    System.Diagnostics.Debug.WriteLine( axTree1.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 AxTree1_AfterExpandItem(ByVal sender As Object, ByVal e As
```

```
AxEXTREELib._ITreeEvents_AfterExpandItemEvent) Handles AxTree1.AfterExpandItem
    Debug.WriteLine(IIf(AxTree1.Items.ExpandItem(e.item), "expanded", "collapsed"))
End Sub
```

The following C++ sample prints the item's state when it is expanded or collapsed:

```
void OnAfterExpandItemTree1(long Item)
{
    CItems items = m_tree.GetItems();
    CString strFormat;
    strFormat.Format( "%s", items.GetExpandItem( Item ) ? "expanded" : "collapsed" );
    OutputDebugString( strFormat );
}
```

The following VFP sample sample prints the item's state when it is expanded or collapsed:

```
*** ActiveX Control Event ***
LPARAMETERS item

with thisform.Tree1.Items
    if ( .ExpandItem(item) )
        wait window "expanded" nowait
    else
        wait window "collapsed" nowait
    endif
endwith
```

event AllowAutoDrag (Item as HITEM, NewParent as HITEM, InsertA as HITEM, InsertB as HITEM, ByRef Cancel as Boolean)

Occurs when the user drags the item between InsertA and InsertB as child of NewParent.

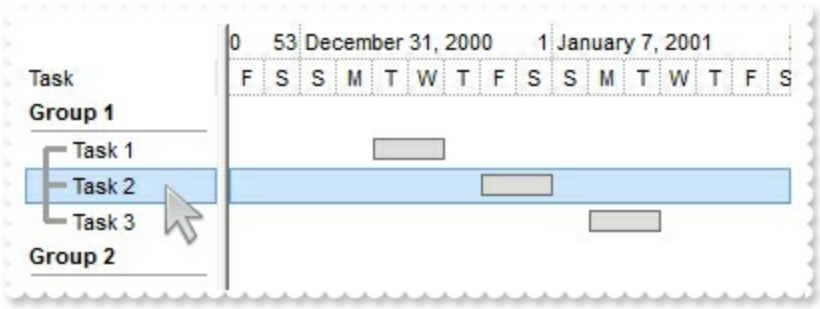
Type	Description
Item as HITEM	A long expression that specifies the handle of the item being dragged.
NewParent as HITEM	A long expression that specifies the handle of the newly parent, to insert the dragging Item. If 0, it indicates root items. The ItemParent property indicates the currently parent of the item.
InsertA as HITEM	A long expression that specifies the handle of the item to insert the dragging Item after. If 0, it indicates that no item after.
InsertB as HITEM	A long expression that specifies the handle of the item to insert the dragging Item before. If 0, it indicates that no item before.
Cancel as Boolean	(By Reference) A Boolean expression that specifies whether the operation can continue (this parameter is by reference)

The AllowAutoDrag event occurs when the user drags the item between InsertA and InsertB as child of NewParent, using the AutoDrag property. The AutoDrag feature indicates what the control does when the user clicks an item and starts dragging it. For instance, using the AutoDrag feature you can let the user arrange the items in the control, or can drop the selection to a any OLE compliant applications like Microsoft Word, Excel and so on... The AllowAutoDrag event may fire when the AutoDrag property is any of the following values:

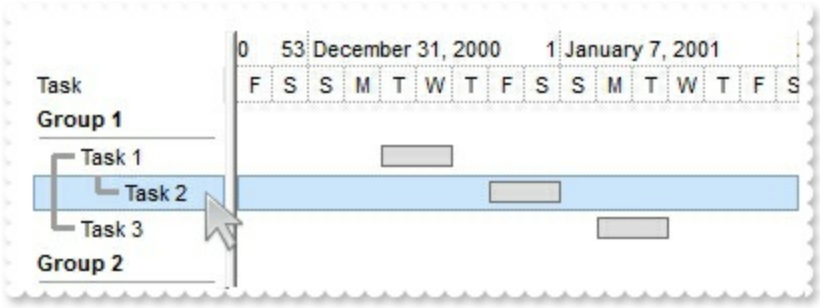
- exAutoDragPosition... (the item can be dragged from a position to another, but not outside of its group)
- exAutoDragPositionKeepIndent... (the item can be dragged to any position or to any parent, while the dragging object keeps its indentation)
- exAutoDragPositionAny... (the item can be dragged to any position or to any parent, with no restriction)

You can use the AllowAutoDrag event to cancel or continue drag and drop operation using the AutoDrag property.

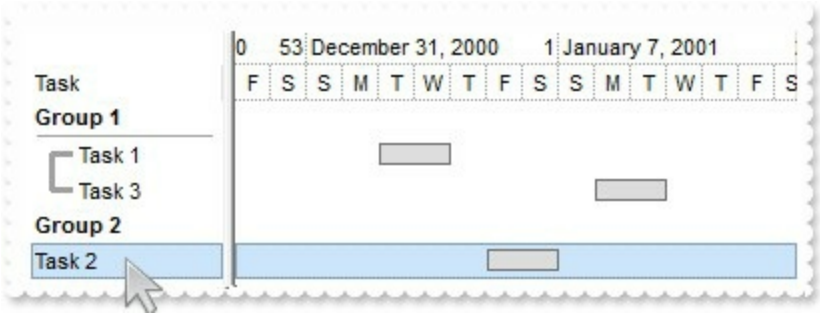
The following screen shot shows the NewParent, InsertA and InsertB parameters, when "Task 2" is dragging to a new position:



- NewParent is "Group 1"
- InsertA is "Task 1"
- InsertB is "Task 3"



- NewParent is "Task 1"
- InsertA is 0
- InsertB is 0



- NewParent is 0
- InsertA is "Group 2"
- InsertB is 0

Syntax for AllowAutoDrag event, **/NET** version, on:

C#

```
private void AllowAutoDrag(object sender,int  Item,int  NewParent,int
InsertA,int  InsertB,ref bool  Cancel)
{
}
```

VB

```
Private Sub AllowAutoDrag(ByVal sender As System.Object,ByVal Item As
Integer,ByVal NewParent As Integer,ByVal InsertA As Integer,ByVal InsertB As
```

```
Integer,ByRef Cancel As Boolean) Handles AllowAutoDrag
End Sub
```

Syntax for AllowAutoDrag event, **/COM** version, on:

C#

```
private void AllowAutoDrag(object sender,
AxEXTREELib._ITreeEvents_AllowAutoDragEvent e)
{
}
```

C++

```
void OnAllowAutoDrag(long  Item,long  NewParent,long  InsertA,long
InsertB,BOOL FAR*  Cancel)
{
}
```

C++ Builder

```
void __fastcall AllowAutoDrag(TObject *Sender,Extreelib_tlb::HITEM
Item,Extreelib_tlb::HITEM  NewParent,Extreelib_tlb::HITEM
InsertA,Extreelib_tlb::HITEM  InsertB,VARIANT_BOOL *  Cancel)
{
}
```

Delphi

```
procedure AllowAutoDrag(ASender: TObject; Item : HITEM;NewParent :
HITEM;InsertA : HITEM;InsertB : HITEM;var Cancel : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AllowAutoDrag(sender: System.Object; e:
AxEXTREELib._ITreeEvents_AllowAutoDragEvent);
begin
end;
```

Powe…

```
begin event AllowAutoDrag(long  Item,long  NewParent,long  InsertA,long
InsertB,boolean  Cancel)

end event AllowAutoDrag
```

VB.NET

```
Private Sub AllowAutoDrag(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_AllowAutoDragEvent) Handles AllowAutoDrag
```

```
End Sub
```

VB6

```
Private Sub AllowAutoDrag(ByVal Item As EXTREELibCtl.HITEM,ByVal NewParent
As EXTREELibCtl.HITEM,ByVal InsertA As EXTREELibCtl.HITEM,ByVal InsertB As
EXTREELibCtl.HITEM,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub AllowAutoDrag(ByVal Item As Long,ByVal NewParent As Long,ByVal
InsertA As Long,ByVal InsertB As Long,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS Item,NewParent,InsertA,InsertB,Cancel


```

Xbas…

```
PROCEDURE OnAllowAutoDrag(oTree,Item,NewParent,InsertA,InsertB,Cancel)

RETURN
```

Syntax for AllowAutoDrag event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AllowAutoDrag(Item,NewParent,InsertA,InsertB,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AllowAutoDrag(Item,NewParent,InsertA,InsertB,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAllowAutoDrag HITEM   llItem HITEM   llNewParent HITEM
llInsertA HITEM   llInsertB Boolean   llCancel
    Forward Send OnComAllowAutoDrag llItem llNewParent llInsertA llInsertB
llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_AllowAutoDrag(Item,NewParent,InsertA,InsertB,Cancel) CLASS
MainDialog
RETURN NIL
```

X++
```
void onEvent_AllowAutoDrag(int _Item,int _NewParent,int _InsertA,int _InsertB,COMVariant /*bool*/ _Cancel)
{
}
```

XBasic
```
function AllowAutoDrag as v (Item as OLE::Exontrol.Tree.1::HITEM,NewParent as OLE::Exontrol.Tree.1::HITEM,InsertA as OLE::Exontrol.Tree.1::HITEM,InsertB as OLE::Exontrol.Tree.1::HITEM,Cancel as L)
end function
```

dBASE
```
function nativeObject_AllowAutoDrag(Item,NewParent,InsertA,InsertB,Cancel)
return
```

The AllowDragDrop event triggers contiguously while the user drags / hovers the focus/selection of items over the control. The GetAsyncKeyState API method can be used to detect whether the mouse button has been released, and so the drop action occurs.

The following VB sample displays "Drag" while user dragging the items, and displays "Drop", when drop operation starts.

```
Private Sub Tree1_AllowAutoDrag(ByVal Item As EXTREELibCtl.HITEM, ByVal NewParent As EXTREELibCtl.HITEM, ByVal InsertA As EXTREELibCtl.HITEM, ByVal InsertB As EXTREELibCtl.HITEM, Cancel As Boolean)
  With Tree1
    Debug.Print "Drag"
    If (GetAsyncKeyState(VK_LBUTTON) = 0) Then
      Debug.Print "Drop"
    End If
  End With
End Sub
```

where declarations for GetAsyncKeyState API used is:

```
Private Const VK_LBUTTON = &H1
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer
```

Once you run the code, you will notice that the AllowAutoDrag event "Drop" may be fired multiple times, so we suggest to postpone any of your actions (like displaying a message box), by posting a window message or use a timer event, to let the control handles /

completes the event as in the following sample:

```
Private Sub Tree1_AllowAutoDrag(ByVal Item As EXTREELibCtl.HITEM, ByVal NewParent As
EXTREELibCtl.HITEM, ByVal InsertA As EXTREELibCtl.HITEM, ByVal InsertB As
EXTREELibCtl.HITEM, Cancel As Boolean)
   With Tree1
      Debug.Print "Drag"
      If (GetAsyncKeyState(VK_LBUTTON) = 0) Then
         mctlTimerDrop.Enabled = True
      End If
   End With
End Sub
```

where mctlTimerDrop is defined as follows:

```
Dim WithEvents mctlTimerDrop As VB.Timer

Private Sub mctlTimerDrop_Timer()
   mctlTimerDrop.Enabled = False
   MsgBox "Drop."
End Sub


Private Sub Form_Load()
   Set mctlTimerDrop = Me.Controls.Add("VB.Timer", "DropTimer1")
   With mctlTimerDrop
      .Enabled = False
      .Interval = 100
   End With
End Sub
```

event AnchorClick (AnchorID as String, Options as String)

Occurs when an anchor element is clicked.

Type	Description
AnchorID as String	A string expression that indicates the identifier of the anchor
Options as String	A string expression that specifies options of the anchor element.

The control fires the AnchorClick event to notify that the user clicks an anchor element. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The **<a>** element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The AnchorClick event is fired only if prior clicking the control it shows the hand cursor. For instance, if the cell is disabled, the hand cursor is not shown when hovers the anchor element, and so the AnchorClick event is not fired. Use the FormatAnchor property to specify the visual effect for anchor elements. For instance, if the user clicks the anchor *<a1>anchor*, the control fires the AnchorClick event, where the AnchorID parameter is 1, and the Options parameter is empty. Also, if the user clicks the anchor *<a 1;yourextradata>anchor*, the AnchorID parameter of the AnchorClick event is 1, and the Options parameter is "yourextradata". Use the AnchorFromPoint property to retrieve the identifier of the anchor element from the cursor.

Syntax for AnchorClick event, **/NET** version, on:

C#

```
private void AnchorClick(object sender,string AnchorID,string Options)
{
}
```

VB

```
Private Sub AnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles AnchorClick
End Sub
```

Syntax for AnchorClick event, **/COM** version, on:

C#

```
private void AnchorClick(object sender,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_AnchorClickEvent);
begin
end;
```

Powe…

```
begin event AnchorClick(string AnchorID,string Options)
end event AnchorClick
```

VB.NET

```
Private Sub AnchorClick(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_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(oTree,AnchorID,Options)
RETURN
```

Syntax for AnchorClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AnchorClick(AnchorID,Options)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AnchorClick(AnchorID,Options)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAnchorClick String llAnchorID String llOptions
    Forward Send OnComAnchorClick llAnchorID llOptions
End_Procedure
```

Visual Objects

```
METHOD OCX_AnchorClick(AnchorID,Options) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AnchorClick(str _AnchorID,str _Options)
{
}
```

XBasic

```
function AnchorClick as v (AnchorID as C,Options as C)
end function
```

dBASE

```
function nativeObject_AnchorClick(AnchorID,Options)
return
```

event BeforeCellEdit (Item as HITEM, ColIndex as Long, ByRef Value as Variant, ByRef Cancel as Variant)

Occurs just before the user enters edit mode by clicking twice in a cell.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being changed.
ColIndex as Long	A long expression that specifies the index of the column where the change occurs, or the handle of the cell being edited if the Item parameter is 0.
Value as Variant	(By Reference) A Variant expression that indicates the edit's caption. By default, the caption of the edit control is the cell's caption. The user can change the text that the edit control displays.
Cancel as Variant	(By Reference) A boolean expression that indicates whether the control cancels the default operation.

The BeforeCellEdit event notifies your application that the user starts editing a cell. Use the Edit method to programmatically edits a cell. Use the AllowEdit property to enable edit feature in the tree control. Use the BeforeCellEdit event to cancel editing cells or to change the edit's caption before it is displayed. Use the AfterCellEdit to change the cell's caption when the edit operation ends. The CancelCellEdit event occurs if the user cancels the edit operation. The CancelCellEdit event is fired when the user presses ESC key while editing or when the user clicks outside the edit field.

Syntax for BeforeCellEdit event, **/NET** version, on:

C#

```
private void BeforeCellEdit(object sender,int Item,int ColIndex,ref object Value,ref object Cancel)
{
}
```

VB

```
Private Sub BeforeCellEdit(ByVal sender As System.Object,ByVal Item As Integer,ByVal ColIndex As Integer,ByRef Value As Object,ByRef Cancel As Object) Handles BeforeCellEdit
End Sub
```

Syntax for BeforeCellEdit event, **/COM** version, on:

C#

```
private void BeforeCellEdit(object sender, AxEXTREELib._ITreeEvents_BeforeCellEditEvent
e)
{
}
```

C++

```
void OnBeforeCellEdit(long Item,long ColIndex,VARIANT FAR* Value,VARIANT
FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall BeforeCellEdit(TObject *Sender,Extreelib_tlb::HITEM Item,long
ColIndex,Variant * Value,Variant * Cancel)
{
}
```

Delphi

```
procedure BeforeCellEdit(ASender: TObject; Item : HITEM;ColIndex : Integer;var
Value : OleVariant;var Cancel : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeCellEdit(sender: System.Object; e:
AxEXTREELib._ITreeEvents_BeforeCellEditEvent);
begin
end;
```

Powe…

```
begin event BeforeCellEdit(long Item,long ColIndex,any Value,any Cancel)
end event BeforeCellEdit
```

VB.NET

```
Private Sub BeforeCellEdit(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_BeforeCellEditEvent) Handles BeforeCellEdit
End Sub
```

VB6

```
Private Sub BeforeCellEdit(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As
Long,Value As Variant,Cancel As Variant)
End Sub
```

VBA

```
Private Sub BeforeCellEdit(ByVal Item As Long,ByVal ColIndex As Long,Value As
Variant,Cancel As Variant)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,Value,Cancel
```

Xbas…

```
PROCEDURE OnBeforeCellEdit(oTree,Item,ColIndex,Value,Cancel)
RETURN
```

Syntax for BeforeCellEdit event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="BeforeCellEdit(Item,ColIndex,Value,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function BeforeCellEdit(Item,ColIndex,Value,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComBeforeCellEdit HITEM llItem Integer llColIndex Variant llValue
Variant llCancel
    Forward Send OnComBeforeCellEdit llItem llColIndex llValue llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_BeforeCellEdit(Item,ColIndex,Value,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_BeforeCellEdit(int _Item,int _ColIndex,COMVariant /*variant*/
_Value,COMVariant /*variant*/ _Cancel)
{
}
```

XBasic

```
function BeforeCellEdit as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as
N,Value as A,Cancel as A)
```

```
end function
```

dBASE

```
function nativeObject_BeforeCellEdit(Item,ColIndex,Value,Cancel)
return
```

The following VB sample cancels editing of any cell that belongs to the first column:

```
Private Sub Tree1_BeforeCellEdit(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long, Value As Variant, Cancel As Variant)
   Cancel = ColIndex = 0
End Sub
```

The following VB.NET sample cancels editing of any cell that belongs to the first column:

```
Private Sub AxTree1_BeforeCellEdit(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_BeforeCellEditEvent) Handles AxTree1.BeforeCellEdit
      e.cancel = e.colIndex = 0
End Sub
```

The following C# sample cancels editing of any cell that belongs to the first column:

```
private void axTree1_BeforeCellEdit(object sender,
AxEXTREELib._ITreeEvents_BeforeCellEditEvent e)
{
   e.cancel = e.colIndex == 0;
}
```

The following C++ sample cancels editing of any cell that belongs to the first column:

```
void OnBeforeCellEditTree1(long Item, long ColIndex, VARIANT FAR* Value, VARIANT FAR*
Cancel)
{
   if ( ColIndex == 0 )
   {
      V_VT( Cancel ) = VT_BOOL;
      V_BOOL( Cancel ) = VARIANT_TRUE;
   }
}
```

The following VFP sample cancels editing of any cell that belongs to the first column:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, value, cancel

if ( colindex = 0 )
   cancel = .t.
endif
```

event BeforeExpandItem (Item as HITEM, ByRef 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	(By Reference) A boolean expression that indicates whether the control cancel expanding or collapsing the item.

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.

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,
AxEXTREELib._ITreeEvents_BeforeExpandItemEvent e)
{
}
```

C++
```
void OnBeforeExpandItem(long Item,VARIANT FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall BeforeExpandItem(TObject *Sender,Extreelib_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:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_BeforeExpandItemEvent) Handles BeforeExpandItem
End Sub
```

VB6

```
Private Sub BeforeExpandItem(ByVal Item As EXTREELibCtl.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(oTree,Item,Cancel)
RETURN
```

Syntax for BeforeExpandItem event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="BeforeExpandItem(Item,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function BeforeExpandItem(Item,Cancel)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComBeforeExpandItem HITEM llItem Variant llCancel
    Forward Send OnComBeforeExpandItem llItem llCancel
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.Tree.1::HITEM,Cancel as A)
end function
```

dBASE
```
function nativeObject_BeforeExpandItem(Item,Cancel)
return
```

The following VB sample cancels expanding or collapsing items:

```
Private Sub Tree1_BeforeExpandItem(ByVal Item As EXTREELibCtl.HITEM, Cancel As Variant)
    Cancel = True
End Sub
```

The following VB sample prints the item's state when it is expanded or collapsed:

```
Private Sub Tree1_AfterExpandItem(ByVal Item As EXTREELibCtl.HITEM)
    Debug.Print "The " & Item & " item was " & IIf(Tree1.Items.ExpandItem(Item),
"expanded", "collapsed")
End Sub
```

The following C# sample cancels expanding or collapsing items:

```
private void axTree1_BeforeExpandItem(object sender,
AxEXTREELib._ITreeEvents_BeforeExpandItemEvent e)
{
   e.cancel = true;
}
```

The following VB.NET sample cancels expanding or collapsing items:

```
Private Sub AxTree1_BeforeExpandItem(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_BeforeExpandItemEvent) Handles AxTree1.BeforeExpandItem
   e.cancel = True
End Sub
```

The following C++ sample cancels expanding or collapsing items:

```
void OnBeforeExpandItemTree1(long Item, VARIANT FAR* Cancel)
{
   V_VT( Cancel ) = VT_BOOL;
   V_BOOL( Cancel ) = VARIANT_TRUE;
}
```

The following VFP sample cancels expanding or collapsing items:

```
*** ActiveX Control Event ***
LPARAMETERS item, cancel

cancel = .t.
```

event CancelCellEdit (Item as HITEM, ColIndex as Long, Reserved as Variant)

Occurs if the edit operation is canceled.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item being edited.
ColIndex as Long	A long expression that indicates the column's index
Reserved as Variant	Contains the caption of the edit control when the user presses the ESC key, or when the user clicks outside the edit field.

The CancelCellEdit event notifies your application that the user cancels the edit operation. The CancelCellEdit event is fired when the user presses ESC key while editing or when the user clicks outside the edit field. The control fires BeforeCellEdit, AfterCellEdit, CancelCellEdit events if the AllowEdit property is True. The AfterCellEdit event notifies your application that the user alters the cell's caption. Use the Edit method to programmatically edit a cell.

Syntax for CancelCellEdit event, **/NET** version, on:

C#
```
private void CancelCellEdit(object sender,int Item,int ColIndex,object Reserved)
{
}
```

VB
```
Private Sub CancelCellEdit(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer,ByVal Reserved As Object) Handles
CancelCellEdit
End Sub
```

Syntax for CancelCellEdit event, **/COM** version, on:

C#
```
private void CancelCellEdit(object sender,
AxEXTREELib._ITreeEvents_CancelCellEditEvent e)
{
}
```

C++
```
void OnCancelCellEdit(long Item,long ColIndex,VARIANT Reserved)
{
}
```

C++ Builder

```
void __fastcall CancelCellEdit(TObject *Sender,Extreelib_tlb::HITEM Item,long ColIndex,Variant Reserved)
{
}
```

Delphi

```
procedure CancelCellEdit(ASender: TObject; Item : HITEM;ColIndex : Integer;Reserved : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CancelCellEdit(sender: System.Object; e: AxEXTREELib._ITreeEvents_CancelCellEditEvent);
begin
end;
```

Powe…

```
begin event CancelCellEdit(long Item,long ColIndex,any Reserved)
end event CancelCellEdit
```

VB.NET

```
Private Sub CancelCellEdit(ByVal sender As System.Object, ByVal e As AxEXTREELib._ITreeEvents_CancelCellEditEvent) Handles CancelCellEdit
End Sub
```

VB6

```
Private Sub CancelCellEdit(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As Long,ByVal Reserved As Variant)
End Sub
```

VBA

```
Private Sub CancelCellEdit(ByVal Item As Long,ByVal ColIndex As Long,ByVal Reserved As Variant)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,Reserved

```

Xbas…

```
PROCEDURE OnCancelCellEdit(oTree,Item,ColIndex,Reserved)
RETURN
```

Syntax for CancelCellEdit event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="CancelCellEdit(Item,ColIndex,Reserved)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function CancelCellEdit(Item,ColIndex,Reserved)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComCancelCellEdit HITEM llItem Integer llColIndex Variant
llReserved
    Forward Send OnComCancelCellEdit llItem llColIndex llReserved
End_Procedure
```

Visual Objects
```
METHOD OCX_CancelCellEdit(Item,ColIndex,Reserved) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_CancelCellEdit(int _Item,int _ColIndex,COMVariant _Reserved)
{
}
```

XBasic
```
function CancelCellEdit as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as
N,Reserved as A)
end function
```

dBASE
```
function nativeObject_CancelCellEdit(Item,ColIndex,Reserved)
return
```

The following VB sample changes the cell's caption when the user clicks outside the edit field:

```
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

Private Sub Tree1_CancelCellEdit(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long, ByVal Reserved As Variant)
    If Not (GetAsyncKeyState(vbKeyEscape) < 0) Then
        With Tree1.Items
```

```
        .CellCaption(Item, ColIndex) = Reserved
    End With
  End If
End Sub
```

The GetAsyncKeyState function determines whether a key is up or down at the time the function is called. The sample changes the selected caption only if the user didn't press ESC key.

The following C++ sample changes the cell's caption when the user clicks outside the edit field:

```
void OnCancelCellEditTree1(long Item, long ColIndex, const VARIANT FAR& Reserved)
{
  if ( !( GetAsyncKeyState( VK_ESCAPE /*27*/ ) < 0 ) )
  {
    CString strNewCaption = V2S( (VARIANT*)&Reserved );
    m_tree.GetItems().SetCellCaption( COleVariant( Item ), COleVariant( ColIndex ),
COleVariant( strNewCaption ) );
  }
}
```

where the V2S function converts a VARIANT expression to a string expression,

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
  if ( pv )
  {
    if ( pv->vt == VT_ERROR )
      return szDefault;

    COleVariant vt;
    vt.ChangeType( VT_BSTR, pv );
    return V_BSTR( &vt );
  }
  return szDefault;
}
```

event CellButtonClick (Item as HITEM, ColIndex as Long)

Fired after the user clicks on the cell of button type.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item where the user clicks the cell's button.
ColIndex as Long	A long expression that specifies the index of the column where the user clicks the cell's button, or a long expression that indicates the handle of the cell being clicked, if the Item parameter is 0.

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.

Syntax for CellButtonClick event, **/NET** version, on:

C#
```
private void CellButtonClick(object sender,int Item,int ColIndex)
{
}
```

VB
```
Private Sub CellButtonClick(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer) Handles CellButtonClick
End Sub
```

Syntax for CellButtonClick event, **/COM** version, on:

C#
```
private void CellButtonClick(object sender,
AxEXTREELib._ITreeEvents_CellButtonClickEvent e)
{
}
```

C++
```
void OnCellButtonClick(long Item,long ColIndex)
{
}
```

C++ Builder
```
void __fastcall CellButtonClick(TObject *Sender,Extreelib_tlb::HITEM Item,long
ColIndex)
```

```
{
}
```

Delphi

```
procedure CellButtonClick(ASender: TObject; Item : HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellButtonClick(sender: System.Object; e:
AxEXTREELib._ITreeEvents_CellButtonClickEvent);
begin
end;
```

Powe…

```
begin event CellButtonClick(long Item,long ColIndex)
end event CellButtonClick
```

VB.NET

```
Private Sub CellButtonClick(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_CellButtonClickEvent) Handles CellButtonClick
End Sub
```

VB6

```
Private Sub CellButtonClick(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As
Long)
End Sub
```

VBA

```
Private Sub CellButtonClick(ByVal Item As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex

```

Xbas…

```
PROCEDURE OnCellButtonClick(oTree,Item,ColIndex)
RETURN
```

Syntax for CellButtonClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="CellButtonClick(Item,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function CellButtonClick(Item,ColIndex)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComCellButtonClick HITEM llItem Integer llColIndex
    Forward Send OnComCellButtonClick llItem llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_CellButtonClick(Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellButtonClick(int _Item,int _ColIndex)
{
}
```

XBasic

```
function CellButtonClick as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_CellButtonClick(Item,ColIndex)
return
```

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 Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
    Tree1.Items.CellHasButton(Item, 0) = True
End Sub

Private Sub Tree1_CellButtonClick(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long)
    MsgBox "The cell of button type has been clicked"
End Sub
```

The following VB.NET sample displays a message when the user clicks a button in the cell:

```
Private Sub AxTree1_CellButtonClick(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_CellButtonClickEvent) Handles AxTree1.CellButtonClick
    MsgBox("The cell of button type has been clicked")
```

```
End Sub
```

The following C# sample displays a message when the user clicks a button in the cell:

```
private void axTree1_CellButtonClick(object sender,
AxEXTREELib._ITreeEvents_CellButtonClickEvent e)
{
    MessageBox.Show("The cell of button type has been clicked");
}
```

The following C++ sample displays a message when the user clicks a button in the cell:

```
void OnCellButtonClickTree1(long Item, long ColIndex)
{
    MessageBox( "The cell of button type has been clicked." );
}
```

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."
```

event CellImageClick (Item as HITEM, ColIndex as Long)

Occurs when the user clicks the cell's icon.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item where the user clicks the cell's icon.
ColIndex as Long	A long expression that indicates the index of the column where the user clicks the cell's icon, or a long expression that indicates the handle of the cell being clicked, if the Item parameter is 0.

The CellImageClick event is fired when user clicks on the cell's image. Use the CellImage property to assign an icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the ItemFromPoint property to determine the index of the icon being clicked, in case the cell displays multiple icons using the CellImages property. Use the CellHasCheckBox or CellHasRadioButton property to assign a check box or a radio button to a cell.

Syntax for CellImageClick event, **/NET** version, on:

C#
```
private void CellImageClick(object sender,int Item,int ColIndex)
{
}
```

VB
```
Private Sub CellImageClick(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer) Handles CellImageClick
End Sub
```

Syntax for CellImageClick event, **/COM** version, on:

C#
```
private void CellImageClick(object sender,
AxEXTREELib._ITreeEvents_CellImageClickEvent e)
{
}
```

C++
```
void OnCellImageClick(long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall CellImageClick(TObject *Sender,Extreelib_tlb::HITEM Item,long ColIndex)
{
}
```

Delphi

```
procedure CellImageClick(ASender: TObject; Item : HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellImageClick(sender: System.Object; e:
AxEXTREELib._ITreeEvents_CellImageClickEvent);
begin
end;
```

Powe…

```
begin event CellImageClick(long Item,long ColIndex)
end event CellImageClick
```

VB.NET

```
Private Sub CellImageClick(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_CellImageClickEvent) Handles CellImageClick
End Sub
```

VB6

```
Private Sub CellImageClick(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As
Long)
End Sub
```

VBA

```
Private Sub CellImageClick(ByVal Item As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex

```

Xbas…

```
PROCEDURE OnCellImageClick(oTree,Item,ColIndex)
RETURN
```

Syntax for CellImageClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="CellImageClick(Item,ColIndex)" LANGUAGE="JScript">
```

```
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function CellImageClick(Item,ColIndex)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComCellImageClick HITEM llItem Integer llColIndex
    Forward Send OnComCellImageClick llItem llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_CellImageClick(Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellImageClick(int _Item,int _ColIndex)
{
}
```

XBasic

```
function CellImageClick as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_CellImageClick(Item,ColIndex)
return
```

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 Tree1_AddItem(ByVal Item As EXTREELibCtl.HITEM)
    Tree1.Items.CellImage(Item, 0) = 1
End Sub

Private Sub Tree1_CellImageClick(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long)
    Tree1.Items.CellImage(Item, ColIndex) = Tree1.Items.CellImage(Item, ColIndex) Mod 2 +
1
End Sub
```

The following VB sample displays the index of icon being clicked:

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim i As HITEM, h As HitTestInfoEnum, c As Long
   With Tree1
      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 changes the cell's icon being clicked:

```
#include "Items.h"
void OnCellImageClickTree1(long Item, long ColIndex)
{
   CItems items = m_tree.GetItems();
   COleVariant vtItem( Item ), vtColumn( ColIndex );
   items.SetCellImage( vtItem , vtColumn , items.GetCellImage( vtItem, vtColumn ) % 2 + 1
);
}
```

The following C# sample changes the cell's icon being clicked:

```
private void axTree1_CellImageClick(object sender,
AxEXTREELib._ITreeEvents_CellImageClickEvent e)
{
   axTree1.Items.set_CellImage( e.item, e.colIndex, axTree1.Items.get_CellImage( e.item,
e.colIndex ) % 2 + 1 );
}
```

The following VB/NET sample changes the cell's icon being clicked:

```
Private Sub AxTree1_CellImageClick(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_CellImageClickEvent) Handles AxTree1.CellImageClick
   With AxTree1.Items
      .CellImage(e.item, e.colIndex) = .CellImage(e.item, e.colIndex) Mod 2 + 1
```

```
    End With
End Sub
```

The following VFP sample changes the cell's icon being clicked:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex

with thisform.Tree1.Items
    .DefaultItem = item
    .CellImage( 0,colindex ) = .CellImage( 0,colindex ) + 1
endwith
```

event CellStateChanged (Item as HITEM, ColIndex as Long)

Fired after cell's state has been changed.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item where the cell's state is changed.
ColIndex as Long	A long expression that indicates the index of the column where the cell's state is changed, or a long expression that indicates the handle of the cell, if the Item parameter is 0.

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 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.

Syntax for CellStateChanged event, **/NET** version, on:

C#
```
private void CellStateChanged(object sender,int Item,int ColIndex)
{
}
```

VB
```
Private Sub CellStateChanged(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer) Handles CellStateChanged
End Sub
```

Syntax for CellStateChanged event, **/COM** version, on:

C#
```
private void CellStateChanged(object sender,
AxEXTREELib._ITreeEvents_CellStateChangedEvent e)
{
}
```

C++
```
void OnCellStateChanged(long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall CellStateChanged(TObject *Sender,Extreelib_tlb::HITEM Item,long ColIndex)
{
}
```

Delphi

```
procedure CellStateChanged(ASender: TObject; Item : HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellStateChanged(sender: System.Object; e:
AxEXTREELib._ITreeEvents_CellStateChangedEvent);
begin
end;
```

Powe…

```
begin event CellStateChanged(long Item,long ColIndex)
end event CellStateChanged
```

VB.NET

```
Private Sub CellStateChanged(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_CellStateChangedEvent) Handles CellStateChanged
End Sub
```

VB6

```
Private Sub CellStateChanged(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex
As Long)
End Sub
```

VBA

```
Private Sub CellStateChanged(ByVal Item As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex

```

Xbas…

```
PROCEDURE OnCellStateChanged(oTree,Item,ColIndex)
RETURN
```

Syntax for CellStateChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="CellStateChanged(Item,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function CellStateChanged(Item,ColIndex)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComCellStateChanged HITEM llItem Integer llColIndex
    Forward Send OnComCellStateChanged llItem llColIndex
End_Procedure
```

Visual Objects
```
METHOD OCX_CellStateChanged(Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_CellStateChanged(int _Item,int _ColIndex)
{
}
```

XBasic
```
function CellStateChanged as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as N)
end function
```

dBASE
```
function nativeObject_CellStateChanged(Item,ColIndex)
return
```

The following VB sample displays a message when the user clicks a check box or a radio button:

```
Private Sub Tree1_CellStateChanged(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long)
    Debug.Print "The cell """ & Tree1.Items.CellCaption(Item, ColIndex) & """ has changed its state. The new state is " & Iif(Tree1.Items.CellState(Item, ColIndes) = 0, "Unchecked", "Checked")
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 OnCellStateChangedTree1(long Item, long ColIndex)
{
    CItems items = m_tree.GetItems();
    COleVariant vtItem( Item ), vtColumn( ColIndex );
    CString strCellCaption = V2S( &items.GetCellCaption( vtItem, vtColumn ) );
    CString strOutput;
    strOutput.Format( "'%s''s checkbox state is %i\r\n", strCellCaption, items.GetCellState(
vtItem, vtColumn ) );
    OutputDebugString( strOutput );
}
```

The following VB.NET sample displays a message when the user clicks a check box or a radio button:

```
Private Sub AxTree1_CellStateChanged(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_CellStateChangedEvent) Handles AxTree1.CellStateChanged
    Debug.WriteLine("The cell """ & AxTree1.Items.CellCaption(e.item, e.colIndex) & """ has
changed its state. The new state is " & IIf(AxTree1.Items.CellState(e.item, e.colIndex) = 0,
"Unchecked", "Checked"))
End Sub
```

The following C# sample outputs a message when the user clicks a check box or a radio button:

```
private void axTree1_CellStateChanged(object sender,
AxEXTREELib._ITreeEvents_CellStateChangedEvent e)
```

```
{
    string strOutput = axTree1.Items.get_CellCaption( e.item, e.colIndex ).ToString();
    strOutput += " state = " + axTree1.Items.get_CellState(e.item, e.colIndex).ToString() ;
    System.Diagnostics.Debug.WriteLine( strOutput );
}
```

The following VFP sample prints a message when the user clicks a check box or a radio button:

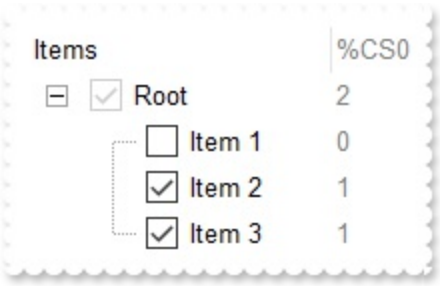
```
*** ActiveX Control Event ***
LPARAMETERS item, colindex

local sOutput
sOutput = ""
with thisform.Tree1.Items
    .DefaultItem = item
    sOutput = .CellCaption( 0, colindex )
    sOutput = sOutput + ", state = " + str(.CellState( 0, colindex ))
    wait window nowait sOutput
endwith
```

event CellStateChanging (Item as HITEM, ColIndex as Long, ByRef NewState as Long)

Fired before cell's state is about to be changed.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item where the cell's state is about to be changed.
ColIndex as Long	A long expression that indicates the index of the column where the cell's state is changed, or a long expression that indicates the handle of the cell, if the Item parameter is 0.
NewState as Long	(By Reference) 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(Item,ColIndex). A cell that contains a radio button or a check box button fires the CellStateChanged event when its state is changed. Use the CellState property to change the cell's state. Use the CellHasRadioButton or CellHasCheckBox property to enable radio or check box button into a cell. Use the Def property to assign check-boxes / radio-buttons for all cells in the column. Use the CellImage property to display an icon in the cell. Use the CellImages property to display multiple icons in the same cell. Use the 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 Tree1_CellStateChanging(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long, NewState As Long)
   With Tree1
      If (.ReadOnly) Then
         With .Items
            NewState = .CellState(Item, ColIndex)
         End With
      End If
   End With
End Sub
```

Syntax for CellStateChanging event, **/NET** version, on:

C#

```
private void CellStateChanging(object sender,int   Item,int   ColIndex,ref int NewState)
{
}
```

VB

```
Private Sub CellStateChanging(ByVal sender As System.Object,ByVal Item As Integer,ByVal ColIndex As Integer,ByRef NewState As Integer) Handles CellStateChanging
End Sub
```

Syntax for CellStateChanging event, **/COM** version, on:

C#

```
private void CellStateChanging(object sender,
AxEXTREELib._ITreeEvents_CellStateChangingEvent e)
{
}
```

C++

```
void OnCellStateChanging(long   Item,long   ColIndex,long FAR*   NewState)
{
}
```

C++ Builder

```
void __fastcall CellStateChanging(TObject *Sender,Extreelib_tlb::HITEM    Item,long ColIndex,long *    NewState)
{
}
```

Delphi

```
procedure CellStateChanging(ASender: TObject; Item : HITEM;ColIndex : Integer;var NewState : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure CellStateChanging(sender: System.Object; e: AxEXTREELib._ITreeEvents_CellStateChangingEvent);
begin
end;
```

Powe…

```
begin event CellStateChanging(long  Item,long  ColIndex,long  NewState)

end event CellStateChanging
```

VB.NET

```
Private Sub CellStateChanging(ByVal sender As System.Object, ByVal e As AxEXTREELib._ITreeEvents_CellStateChangingEvent) Handles CellStateChanging
End Sub
```

VB6

```
Private Sub CellStateChanging(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As Long,NewState As Long)
End Sub
```

VBA

```
Private Sub CellStateChanging(ByVal Item As Long,ByVal ColIndex As Long,NewState As Long)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,NewState

```

Xbas…

```
PROCEDURE OnCellStateChanging(oTree,Item,ColIndex,NewState)

RETURN
```

Syntax for CellStateChanging event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="CellStateChanging(Item,ColIndex,NewState)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function CellStateChanging(Item,ColIndex,NewState)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComCellStateChanging HITEM   llItem Integer   llColIndex Integer
llNewState
    Forward Send OnComCellStateChanging llItem llColIndex llNewState
End_Procedure
```

Visual Objects

```
METHOD OCX_CellStateChanging(Item,ColIndex,NewState) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_CellStateChanging(int   _Item,int   _ColIndex,COMVariant /*long*/
_NewState)
{
}
```

XBasic

```
function CellStateChanging as v (Item  as  OLE::Exontrol.Tree.1::HITEM,ColIndex  as
N,NewState  as  N)
end function
```

dBASE

```
function nativeObject_CellStateChanging(Item,ColIndex,NewState)
return
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the control.

Type	Description

The Click event is fired when the user releases the left mouse button over the control. Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Syntax for Click event, **/NET** version, on:

C#

```
private void Click(object sender)
{
}
```

VB

```
Private Sub Click(ByVal sender As System.Object) Handles Click
End Sub
```

Syntax for Click event, **/COM** version, on:

C#

```
private void ClickEvent(object sender, EventArgs e)
{
}
```

C++

```
void OnClick()
{
}
```

C++ Builder

```
void __fastcall Click(TObject *Sender)
{
}
```

Delphi

```
procedure Click(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure ClickEvent(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event Click()
end event Click
```

VB.NET

```
Private Sub ClickEvent(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClickEvent
End Sub
```

VB6

```
Private Sub Click()
End Sub
```

VBA

```
Private Sub Click()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnClick(oTree)
RETURN
```

Syntax for Click event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Click()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Click()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComClick
    Forward Send OnComClick
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_Click() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Click()
{
}
```

XBasic

```
function Click as v ()
end function
```

dBASE

```
function nativeObject_Click()
return
```

event ColumnClick (Column as Column)

Fired after the user clicks on column's header.

Type	Description
Column as Column	A Column object that indicates clicked column.

The ColumnClick event is fired when the user clicks the column's header. By default, the control sorts by the column when user clicks the column's header. Use the SortOnClick property to specify the operation that control does when user clicks the column's caption. Use the ColumnFromPoint property to access the column from point. Use the ItemFromPoint property to access the item from point. The control fires Sort method when the control sorts a column. Use the MouseDown or MouseUp event to notify the control when the user clicks the control, including the columns.

Syntax for ColumnClick event, **/NET** version, on:

C#
```
private void ColumnClick(object sender,exontrol.EXTREELib.Column Column)
{
}
```

VB
```
Private Sub ColumnClick(ByVal sender As System.Object,ByVal Column As
exontrol.EXTREELib.Column) Handles ColumnClick
End Sub
```

Syntax for ColumnClick event, **/COM** version, on:

C#
```
private void ColumnClick(object sender,
AxEXTREELib._ITreeEvents_ColumnClickEvent e)
{
}
```

C++
```
void OnColumnClick(LPDISPATCH Column)
{
}
```

C++ Builder
```
void __fastcall ColumnClick(TObject *Sender,Extreelib_tlb::IColumn *Column)
{
}
```

Delphi

```
procedure ColumnClick(ASender: TObject; Column : IColumn);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ColumnClick(sender: System.Object; e:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_ColumnClickEvent) Handles ColumnClick
End Sub
```

VB6

```
Private Sub ColumnClick(ByVal Column As EXTREELibCtl.IColumn)
End Sub
```

VBA

```
Private Sub ColumnClick(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnColumnClick(oTree,Column)
RETURN
```

Syntax for ColumnClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="ColumnClick(Column)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ColumnClick(Column)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComColumnClick Variant llColumn
    Forward Send OnComColumnClick llColumn
End_Procedure
```

Visual Objects

```
METHOD OCX_ColumnClick(Column) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ColumnClick(COM _Column)
{
}
```

XBasic

```
function ColumnClick as v (Column as OLE::Exontrol.Tree.1::IColumn)
end function
```

dBASE

```
function nativeObject_ColumnClick(Column)
return
```

The following VB sample displays the caption of the column being clicked:

```
Private Sub Tree1_ColumnClick(ByVal Column As EXTREELibCtl.IColumn)
    Debug.Print Column.Caption
End Sub
```

The following C++ sample displays the caption of the column being clicked:

```
#include "Column.h"
void OnColumnClickTree1(LPDISPATCH Column)
{
    CColumn column( Column );
    column.m_bAutoRelease = FALSE;
    MessageBox( column.GetCaption() );

}
```

The following VB.NET sample displays the caption of the column being clicked:

```
Private Sub AxTree1_ColumnClick(ByVal sender As Object, ByVal e As
```

```
AxEXTREELib._ITreeEvents_ColumnClickEvent) Handles AxTree1.ColumnClick
    MessageBox.Show(e.column.Caption)
End Sub
```

The following C# sample displays the caption of the column being clicked:

```
private void axTree1_ColumnClick(object sender,
AxEXTREELib._ITreeEvents_ColumnClickEvent e)
{
    MessageBox.Show( e.column.Caption );
}
```

The following VFP sample displays the caption of the column being clicked:

```
*** ActiveX Control Event ***
LPARAMETERS column

with column
    wait window nowait .Caption
endwith
```

event DblClick (Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user dblclk the left mouse button over an object.

Type	Description
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The DblClick event is fired when user double clicks the control. Use the ItemFromPoint method to determine the cell over the cursor. Use the ExpandOnDblClk property to specify whether an item is expanded or collapsed when user double clicks it. Use the ColumnFromPoint property to get the column from point.

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, AxEXTREELib._ITreeEvents_DblClickEvent e)
{
}
```

C++

```
void OnDblClick(short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall DblClick(TObject *Sender,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure DblClick(ASender: TObject; Shift : Smallint;X : Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure DblClick(sender: System.Object; e:
AxEXTREELib._ITreeEvents_DblClickEvent);
begin
end;
```

Powe…

```
begin event DblClick(integer Shift,long X,long Y)
end event DblClick
```

VB.NET

```
Private Sub DblClick(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_DblClickEvent) Handles DblClick
End Sub
```

VB6

```
Private Sub DblClick(Shift As Integer,X As Single,Y As Single)
End Sub
```

VBA

```
Private Sub DblClick(ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Shift,X,Y

```

Xbas…

```
PROCEDURE OnDblClick(oTree,Shift,X,Y)
RETURN
```

Syntax for DblClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="DblClick(Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function DblClick(Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComDblClick Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY
    Forward Send OnComDblClick llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_DblClick(Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_DblClick(int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function DblClick as v (Shift as N,X as OLE::Exontrol.Tree.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Tree.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_DblClick(Shift,X,Y)
return
```

The following VB sample prints a message when an item has been double clicked:

```
Private Sub Tree1_DblClick(Shift As Integer, X As Single, Y As Single)
   ' Converts the container coordinates to client coordinates
   X = X / Screen.TwipsPerPixelX
   Y = Y / Screen.TwipsPerPixelY
   Dim h As HITEM
   Dim c As Long, hit as Long
   ' Gets the item from (X,Y)
   h = Tree1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Then
      MsgBox "The " & h & " item has been double clicked."
   End If
```

```
End Sub
```

The following VB sample displays a message when a cell has been double clicked:

```
Private Sub Tree1_DblClick(Shift As Integer, X As Single, Y As Single)
   ' Converts the container coordinates to client coordinates
   X = X / Screen.TwipsPerPixelX
   Y = Y / Screen.TwipsPerPixelY
   Dim h As HITEM
   Dim c As Long, hit as Long
   ' Gets the item from (X,Y)
   h = Tree1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Then
      MsgBox "The """ & Tree1.Items.CellCaption(h, c) & """ cell has been double clicked."
   End If
End Sub
```

The following C++ sample displays the caption of the cell being double clicked (including the inner cells):

```
#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 OnDblClickTree1(short Shift, long X, long Y)
{
```

```
    long c = NULL, hit = NULL;
    long h = m_tree.GetItemFromPoint( X, Y, &c, &hit );
    if ( ( h != 0 ) || ( c != 0 ) )
    {
        COleVariant vtItem( h ), vtColumn( c );
        CString strCaption = V2S( &m_tree.GetItems().GetCellCaption( vtItem, vtColumn ) );
        MessageBox( strCaption );
    }
}
```

The following VB.NET sample displays the caption of the cell being double clicked (including the inner cells):

```
Private Sub AxTree1_DblClick(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_DblClickEvent) Handles AxTree1.DblClick
    Dim h As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
    With AxTree1
        h = .get_ItemFromPoint(e.x, e.y, c, hit)
        If Not (h = 0) Or Not (c = 0) Then
            MessageBox.Show(.Items.CellCaption(h, c))
        End If
    End With
End Sub
```

The following C# sample displays the caption of the cell being double clicked (including the inner cells):

```
private void axTree1_DblClick(object sender, AxEXTREELib._ITreeEvents_DblClickEvent e)
{
    EXTREELib.HitTestInfoEnum hit;
    int c = 0, h = axTree1.get_ItemFromPoint( e.x, e.y, out c, out hit );
    if ( ( h != 0 ) || ( c != 0 ) )
        MessageBox.Show( axTree1.Items.get_CellCaption( h, c ).ToString() );
}
```

The following VFP sample displays the caption of the cell being double clicked:

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

```
local c, hit
c = 0
hit = 0

with thisform.Tree1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem != 0 )
      wait window nowait .Items.CellCaption( 0, c )
   endif
endwith
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Use the EventParam(-2) to display entire information about fired event (such as name, identifier, and properties).

The Event notification occurs ANY time the control fires an event.

This is useful for X++ language, which does not support event with parameters passed by reference.

In X++ the "Error executing code: FormActiveXControl (data source), method ... called with invalid parameters" occurs when handling events that have parameters passed by reference. Passed by reference, means that in the event handler, you can change the value for that parameter, and so the control will takes the new value, and use it. The X++ is NOT able to handle properly events with parameters by reference, so we have the solution.

The solution is using and handling the Event notification and EventParam method., instead handling the event that gives the "invalid parameters" error executing code.

Let's presume that we need to handle the BarParentChange event to change the _Cancel parameter from false to true, which fires the "Error executing code: FormActiveXControl (data source), method onEvent_BarParentChange called with invalid parameters." We need to know the identifier of the BarParentChange event (each event has an unique identifier and it is static, defined in the control's type library). If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   print 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._ITreeEvents_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._ITreeEvents_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._ITreeEvents_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(oTree,EventID)
RETURN
```

Syntax for Event event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Event(EventID)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Event(EventID)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEvent Integer llEventID
    Forward Send OnComEvent llEventID
End_Procedure
```

Visual Objects

```
METHOD OCX_Event(EventID) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Event(int _EventID)
{
}
```

XBasic

```
function Event as v (EventID as N)
end function
```

dBASE

```
function nativeObject_Event(EventID)
return
```

event FilterChange ()

Occurs when filter was changed.

Type	Description

Use the FilterChange event to notify your application that the control's filter is changed. The FilterChanging event occurs just before applying the filter. Use the Filter and FilterType properties to retrieve the column's filter string, if case, and the column's filter type. The ApplyFilter and ClearFilter methods fire the FilterChange event. Use the DisplayFilterButton property to add a filter bar button to the column's caption. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar. Use the VisibleItemCount property to retrieve the number of filtered items. Use the FilterBarCaption property to change the caption in the filter bar once a new filter is applied. The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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.

Syntax for FilterChange event, **/NET** version, on:

C#
```
private void FilterChange(object sender)
{
}
```

VB
```
Private Sub FilterChange(ByVal sender As System.Object) Handles FilterChange
End Sub
```

Syntax for FilterChange event, **/COM** version, on:

C#
```
private void FilterChange(object sender, EventArgs e)
{
}
```

C++
```
void OnFilterChange()
{
}
```

C++ Builder
```
void __fastcall FilterChange(TObject *Sender)
{
}
```

Delphi
```
procedure FilterChange(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)
```
procedure FilterChange(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…
```
begin event FilterChange()
end event FilterChange
```

VB.NET
```
Private Sub FilterChange(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FilterChange
End Sub
```

VB6
```
Private Sub FilterChange()
End Sub
```

VBA
```
Private Sub FilterChange()
End Sub
```

VFP
```
LPARAMETERS nop

```

Xbas…
```
PROCEDURE OnFilterChange(oTree)
RETURN
```

Syntax for FilterChange event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="FilterChange()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function FilterChange()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComFilterChange
    Forward Send OnComFilterChange
End_Procedure
```

Visual Objects

```
METHOD OCX_FilterChange() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_FilterChange()
{
}
```

XBasic

```
function FilterChange as v ()
end function
```

dBASE

```
function nativeObject_FilterChange()
return
```

event FilterChanging ()

Notifies your application that the filter is about to change.

Type	Description

The FilterChanging event occurs just before applying the filter. The FilterChange event occurs once the filter is applied, so the list gets filtered. Use the Filter and FilterType properties to retrieve the column's filter string, if case, and the column's filter type. The ApplyFilter and ClearFilter methods fire the FilterChange event. Use the DisplayFilterButton property to add a filter bar button to the column's caption. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar. For instance, you can use the FilterChanging event to start a timer, and count the time to get the filter applied, when the FilterChange event is fired. The FilterBarPromptVisible property specifies whether the filter prompt is visible or hidden. 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.

Syntax for FilterChanging event, **/NET** version, on:

C#
```
private void FilterChanging(object sender)
{
}
```

VB
```
Private Sub FilterChanging(ByVal sender As System.Object) Handles
FilterChanging
End Sub
```

Syntax for FilterChanging event, **/COM** version, on:

C#
```
private void FilterChanging(object sender, EventArgs e)
{
}
```

C++
```
void OnFilterChanging()
{
}
```

C++ Builder
```
void __fastcall FilterChanging(TObject *Sender)
{
}
```

Delphi

```
procedure FilterChanging(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure FilterChanging(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event FilterChanging()
end event FilterChanging
```

VB.NET

```
Private Sub FilterChanging(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FilterChanging
End Sub
```

VB6

```
Private Sub FilterChanging()
End Sub
```

VBA

```
Private Sub FilterChanging()
End Sub
```

VFP

```
LPARAMETERS nop

```

Xbas…

```
PROCEDURE OnFilterChanging(oTree)
RETURN
```

Syntax for FilterChanging event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="FilterChanging()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function FilterChanging()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComFilterChanging
    Forward Send OnComFilterChanging
End_Procedure
```

Visual Objects

```
METHOD OCX_FilterChanging() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_FilterChanging()
{
}
```

XBasic

```
function FilterChanging as v ()
end function
```

dBASE

```
function nativeObject_FilterChanging()
return
```

event FormatColumn (Item as HITEM, ColIndex as Long, ByRef 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	(By Reference) 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).

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,
AxEXTREELib._ITreeEvents_FormatColumnEvent e)
{
}
```

C++

```
void OnFormatColumn(long Item,long ColIndex,VARIANT FAR* Value)
{
}
```

C++ Builder

```
void __fastcall FormatColumn(TObject *Sender,Extreelib_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: AxEXTREELib._ITreeEvents_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 AxEXTREELib._ITreeEvents_FormatColumnEvent) Handles FormatColumn
End Sub
```

VB6

```
Private Sub FormatColumn(ByVal Item As EXTREELibCtl.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(oTree,Item,ColIndex,Value)
RETURN
```

Syntax for FormatColumn event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="FormatColumn(Item,ColIndex,Value)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function FormatColumn(Item,ColIndex,Value)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComFormatColumn HITEM llItem Integer llColIndex Variant llValue
    Forward Send OnComFormatColumn llItem llColIndex llValue
End_Procedure
```

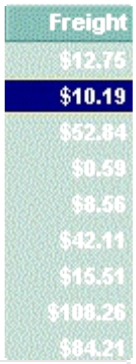
Visual Objects
```
METHOD OCX_FormatColumn(Item,ColIndex,Value) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_FormatColumn(int _Item,int _ColIndex,COMVariant /*variant*/
_Value)
{
}
```

XBasic
```
function FormatColumn as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as
N,Value as A)
end function
```

dBASE
```
function nativeObject_FormatColumn(Item,ColIndex,Value)
return
```

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.



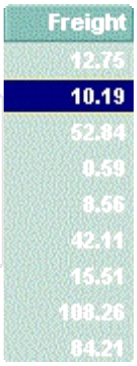

```
Tree1.Columns("Freight").FireFormatColumn = True
Tree1.Columns("Freight").HeaderBold = True
Tree1.Columns("Freight").Alignment = RightAlignment
```

```
Private Sub Tree1_FormatColumn(ByVal Item As EXTREELibCtl.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:




```
Tree1.Columns("Freight").FireFormatColumn = False
Tree1.Columns("Freight").HeaderBold = True
Tree1.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 Tree1_FormatColumn(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long, Value As Variant)
   Value = IIf(Value < 50, "Yes", "No")
End Sub
```

The following VB sample displays the result of adding (concatenating) of two cells:

```
Private Sub Tree1_FormatColumn(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As Long, Value As Variant)
   With Tree1.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 OnFormatColumnTree1(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 AxTree1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_FormatColumnEvent) Handles AxTree1.FormatColumn
    e.value = DateTime.Parse(e.value).ToLongDateString()
End Sub
```

The following C# sample displays a date column using LongDate format:

```
private void axTree1_FormatColumn(object sender,
AxEXTREELib._ITreeEvents_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.Tree1.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.

event HyperLinkClick (Item as HITEM, ColIndex as Long)

Occurs when the user clicks on a hyperlink cell.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
ColIndex as Long	A long expression that indicates the column's index.

The HyperLinkClick event is fired when user clicks a hyperlink cell. A hyperlink cell has the CellHyperLink property on True. The control changes the shape of the cursor when the mouse hovers a hyper linkcell. Use the HyperLinkClick event to notify your application that a hyperlink cell is clicked. Use the HyperLinkColor property to specify the hyperlink color. The HyperLinkClick event is fired only if the user clicks a cell that has the CellHyperLink property on True. Use the ItemFromPoint property to get an item or a cell from point. Use the ColumnFromPoint property to get the column from point.

Syntax for HyperLinkClick event, **/NET** version, on:

C#

```
private void HyperLinkClick(object sender,int Item,int ColIndex)
{
}
```

VB

```
Private Sub HyperLinkClick(ByVal sender As System.Object,ByVal Item As
Integer,ByVal ColIndex As Integer) Handles HyperLinkClick
End Sub
```

Syntax for HyperLinkClick event, **/COM** version, on:

C#

```
private void HyperLinkClick(object sender,
AxEXTREELib._ITreeEvents_HyperLinkClickEvent e)
{
}
```

C++

```
void OnHyperLinkClick(long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall HyperLinkClick(TObject *Sender,Extreelib_tlb::HITEM Item,long
ColIndex)
{
```

```
}
```

Delphi

```
procedure HyperLinkClick(ASender: TObject; Item : HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure HyperLinkClick(sender: System.Object; e:
AxEXTREELib._ITreeEvents_HyperLinkClickEvent);
begin
end;
```

Powe…

```
begin event HyperLinkClick(long Item,long ColIndex)
end event HyperLinkClick
```

VB.NET

```
Private Sub HyperLinkClick(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_HyperLinkClickEvent) Handles HyperLinkClick
End Sub
```

VB6

```
Private Sub HyperLinkClick(ByVal Item As EXTREELibCtl.HITEM,ByVal ColIndex As
Long)
End Sub
```

VBA

```
Private Sub HyperLinkClick(ByVal Item As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex

```

Xbas…

```
PROCEDURE OnHyperLinkClick(oTree,Item,ColIndex)
RETURN
```

Syntax for HyperLinkClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="HyperLinkClick(Item,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function HyperLinkClick(Item,ColIndex)
```

```
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComHyperLinkClick HITEM llItem Integer llColIndex
    Forward Send OnComHyperLinkClick llItem llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_HyperLinkClick(Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_HyperLinkClick(int _Item,int _ColIndex)
{
}
```

XBasic

```
function HyperLinkClick as v (Item as OLE::Exontrol.Tree.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_HyperLinkClick(Item,ColIndex)
return
```

The following VB sample displays the caption of the hyperlink cell that's been clicked:

```
Private Sub Tree1_HyperLinkClick(ByVal Item As EXTREELibCtl.HITEM, ByVal ColIndex As
Long)
   Debug.Print Tree1.Items.CellCaption(Item, ColIndex)
End Sub
```

The following VC sample displays the caption of the hyperlink cell that's been clicked:

```
#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 OnHyperLinkClickTree1(long Item, long ColIndex)
{
    CItems items = m_tree.GetItems();
    COleVariant vtItem( Item ), vtColumn( ColIndex );
    OutputDebugString( V2S( &items.GetCellCaption( vtItem, vtColumn ) ) );
}
```

The following VB.NET sample displays the caption of the hyperlink cell that's been clicked:

```
Private Sub AxTree1_HyperLinkClick(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_HyperLinkClickEvent) Handles AxTree1.HyperLinkClick
    With AxTree1.Items
        Debug.WriteLine(.CellCaption(e.item, e.colIndex))
    End With
End Sub
```

The following C# sample displays the caption of the hyperlink cell that's been clicked:

```
private void axTree1_HyperLinkClick(object sender,
AxEXTREELib._ITreeEvents_HyperLinkClickEvent e)
{
    System.Diagnostics.Debug.WriteLine( axTree1.Items.get_CellCaption(e.item, e.colIndex )
);
}
```

The following VFP sample displays the caption of the hyperlink cell that's been clicked:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex

with thisform.Tree1.Items
    .DefaultItem = item
```

```
    wait window nowait .CellCaption( 0, colindex )
endwith
```

event ItemOleEvent (Item as HITEM, Ev as OleEvent)

Fired when an ActiveX control hosted by an item has fired an event.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item that hosts an ActiveX control.
Ev as OleEvent	An OleEvent object that contains information about the fired event.

The Exontrol's ExTree control supports ActiveX hosting. The InsertItemControl method inserts an item that hosts an ActiveX control. The ItemOleEvent event notifies your application that a hosted ActiveX control fires an event. The ItemObject property gets the ActiveX object hosted by an item that is inserted using the InsertControlItem method. The ItemObject property gets nothing if the item doesn't host an ActiveX control, or if inserting an ActiveX control failed).

Syntax for ItemOleEvent event, **/NET** version, on:

C#
```
private void ItemOleEvent(object sender,int Item,exontrol.EXTREELib.OleEvent Ev)
{
}
```

VB
```
Private Sub ItemOleEvent(ByVal sender As System.Object,ByVal Item As
Integer,ByVal Ev As exontrol.EXTREELib.OleEvent) Handles ItemOleEvent
End Sub
```

Syntax for ItemOleEvent event, **/COM** version, on:

C#
```
private void ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
}
```

C++
```
void OnItemOleEvent(long Item,LPDISPATCH Ev)
{
}
```

C++ Builder
```
void __fastcall ItemOleEvent(TObject *Sender,Extreelib_tlb::HITEM
Item,Extreelib_tlb::IOleEvent *Ev)
```

```
{
}
```

Delphi

```
procedure ItemOleEvent(ASender: TObject; Item : HITEM;Ev : IOleEvent);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ItemOleEvent(sender: System.Object; e:
AxEXTREELib._ITreeEvents_ItemOleEventEvent);
begin
end;
```

Powe…

```
begin event ItemOleEvent(long Item,oleobject Ev)
end event ItemOleEvent
```

VB.NET

```
Private Sub ItemOleEvent(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles ItemOleEvent
End Sub
```

VB6

```
Private Sub ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM,ByVal Ev As
EXTREELibCtl.IOleEvent)
End Sub
```

VBA

```
Private Sub ItemOleEvent(ByVal Item As Long,ByVal Ev As Object)
End Sub
```

VFP

```
LPARAMETERS Item,Ev

```

Xbas…

```
PROCEDURE OnItemOleEvent(oTree,Item,Ev)
RETURN
```

Syntax for ItemOleEvent event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="ItemOleEvent(Item,Ev)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function ItemOleEvent(Item,Ev)
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComItemOleEvent HITEM llItem Variant llEv
    Forward Send OnComItemOleEvent llItem llEv
End_Procedure
```

Visual Objects

```
METHOD OCX_ItemOleEvent(Item,Ev) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ItemOleEvent(int _Item,COM _Ev)
{
}
```

XBasic

```
function ItemOleEvent as v (Item as OLE::Exontrol.Tree.1::HITEM,Ev as
OLE::Exontrol.Tree.1::IOleEvent)
end function
```

dBASE

```
function nativeObject_ItemOleEvent(Item,Ev)
return
```

The following VB sample adds an item that hosts the Microsoft Calendar Control and prints each event fired by that ActiveX control:

```
Tree1.Items.ItemHeight(Tree1.Items.InsertControlItem(, "MSCal.Calendar")) = 256
```


```
Private Sub Tree1_ItemOleEvent(ByVal Item As EXTREELibCtl.HITEM, ByVal Ev As
EXTREELibCtl.IOleEvent)
  Debug.Print "Event name:" & Ev.Name
  If (Ev.CountParam = 0) Then
```

```
    Debug.Print "The event has no arguments."
  Else
    Debug.Print "The event has the following arguments:"
    Dim i As Long
    For i = 0 To Ev.CountParam - 1
      Debug.Print Ev(i).Name; " = " & Ev(i).Value
    Next
  End If
End Sub
```

The following VC sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

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 OnItemOleEventTree1(long Item, LPDISPATCH Ev)
{
   EXTREELib::IOleEventPtr spEvent( Ev );
   CString strOutput;
   strOutput.Format( "Event's name: %s\n", spEvent->Name.operator const char *() );
   OutputDebugString( strOutput );
   if ( spEvent->CountParam == 0 )
      OutputDebugString( "The event has no parameters." );
   else
```

```
    {
        for ( long i = 0; i < spEvent->CountParam; i++ )
        {
            EXTREELib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
            strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
            OutputDebugString( strOutput );
        }
    }
    OutputDebugString( "" );
}
```

The #import clause is required to get the wrapper classes for IOleEvent and IOleEventParam objects, that are not defined by the MFC class wizard. The same #import statement defines the EXTREELib namespace that include all objects and types of the control's TypeLibrary. In case your extree.dll library is located to another place than the system folder or well known path, the path to the library should be provided, in order to let the VC finds the type library.

The following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxTree1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_ItemOleEventEvent) Handles AxTree1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXTREELib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
private void axTree1_ItemOleEvent(object sender,
AxEXTREELib._ITreeEvents_ItemOleEventEvent e)
{
```

```
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXTREELib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following VFP sample displays the events that an ActiveX control fires when it is hosted by an item:

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

event KeyDown (ByRef KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	(By Reference) An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. Use the ExpandOnKeys property to specify whether the user expands or collapses the focused items using arrow keys. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

Syntax for KeyDown event, **/NET** version, on:

C#
```
private void KeyDown(object sender,ref short KeyCode,short Shift)
{
}
```

VB
```
Private Sub KeyDown(ByVal sender As System.Object,ByRef KeyCode As
Short,ByVal Shift As Short) Handles KeyDown
End Sub
```

Syntax for KeyDown event, **/COM** version, on:

C#

```
private void KeyDownEvent(object sender,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_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(oTree,KeyCode,Shift)
RETURN
```

Syntax for KeyDown event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyDown(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyDown(KeyCode,Shift)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyDown Short llKeyCode Short llShift
    Forward Send OnComKeyDown llKeyCode llShift
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyDown(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyDown(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic

```
function KeyDown as v (KeyCode as N,Shift as N)
end function
```

dBASE

```
function nativeObject_KeyDown(KeyCode,Shift)
return
```

event KeyPress (ByRef KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	(By Reference) An integer that returns a standard numeric ANSI keycode.

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

Syntax for KeyPress event, **/NET** version, on:

C#

```
private void KeyPress(object sender,ref short KeyAscii)
{
}
```

VB

```
Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

Syntax for KeyPress event, **/COM** version, on:

C#

```
private void KeyPressEvent(object sender,
AxEXTREELib._ITreeEvents_KeyPressEvent e)
{
}
```

C++

```
void OnKeyPress(short FAR* KeyAscii)
{
}
```

C++ Builder

```
void __fastcall KeyPress(TObject *Sender,short * KeyAscii)
{
}
```

Delphi

```
procedure KeyPress(ASender: TObject; var KeyAscii : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure KeyPressEvent(sender: System.Object; e:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_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(oTree,KeyAscii)
RETURN
```

Syntax for KeyPress event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyPress(KeyAscii)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyPress Short llKeyAscii
    Forward Send OnComKeyPress llKeyAscii
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyPress(KeyAscii) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyPress(COMVariant /*short*/ _KeyAscii)
{
}
```

XBasic

```
function KeyPress as v (KeyAscii as N)
end function
```

dBASE

```
function nativeObject_KeyPress(KeyAscii)
return
```

event KeyUp (ByRef KeyCode as Integer, Shift as Integer)

Occurs when the user releases a key while an object has the focus.

Type	Description
KeyCode as Integer	(By Reference) An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use the KeyUp event procedure to respond to the releasing of a key.

Syntax for KeyUp event, **/NET** version, on:

C#

```
private void KeyUp(object sender,ref short KeyCode,short Shift)
{
}
```

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, AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_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(oTree,KeyCode,Shift)
RETURN
```

Syntax for KeyUp event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyUp(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyUp(KeyCode,Shift)
End Function
```

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyUp Short llKeyCode Short llShift
    Forward Send OnComKeyUp llKeyCode llShift
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyUp(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyUp(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic

```
function KeyUp as v (KeyCode as N,Shift as N)
end function
```

dBASE

```
function nativeObject_KeyUp(KeyCode,Shift)
return
```

event LayoutChanged ()

Occurs when column's position or column's size is changed.

Type	Description

The LayoutChanged event is fired each time when the user resizes a column, or drags the column to a new position. Use the LayoutChanged event to notify your application that the columns position or size is changed. Use the LayoutChanged event to save the columns position and size for future use. Use the Width property to retrieve the column's width. Use the Position property to retrieve the column's position. The Visible property specifies whether a column is shown or hidden. Use the ColumnAutoResize property to specify whether the visible columns fit the control's client area.

Syntax for LayoutChanged event, **/NET** version, on:

C#
```
private void LayoutChanged(object sender)
{
}
```

VB
```
Private Sub LayoutChanged(ByVal sender As System.Object) Handles
LayoutChanged
End Sub
```

Syntax for LayoutChanged event, **/COM** version, on:

C#
```
private void LayoutChanged(object sender, EventArgs e)
{
}
```

C++
```
void OnLayoutChanged()
{
}
```

C++ Builder
```
void __fastcall LayoutChanged(TObject *Sender)
{
}
```

Delphi
```
procedure LayoutChanged(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure LayoutChanged(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event LayoutChanged()
end event LayoutChanged
```

VB.NET

```
Private Sub LayoutChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LayoutChanged
End Sub
```

VB6

```
Private Sub LayoutChanged()
End Sub
```

VBA

```
Private Sub LayoutChanged()
End Sub
```

VFP

```
LPARAMETERS nop

```

Xbas…

```
PROCEDURE OnLayoutChanged(oTree)
RETURN
```

Syntax for LayoutChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="LayoutChanged()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function LayoutChanged()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComLayoutChanged
    Forward Send OnComLayoutChanged
End_Procedure
```

Visual Objects

```
METHOD OCX_LayoutChanged() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_LayoutChanged()
{
}
```

XBasic

```
function LayoutChanged as v ()
end function
```

dBASE

```
function nativeObject_LayoutChanged()
return
```

event MouseDown (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user presses a mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the ItemFromPoint property to get the item from point. Use the ColumnFromPoint property to get the column from point.

Syntax for MouseDown event, **/NET** version, on:

C#
```
private void MouseDownEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub MouseDownEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseDownEvent
End Sub
```

Syntax for MouseDown event, **/COM** version, on:

C#
```
private void MouseDownEvent(object sender,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_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(oTree,Button,Shift,X,Y)
RETURN
```

Syntax for MouseDown event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="MouseDown(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function MouseDown(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComMouseDown Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseDown llButton llShift llX llY
End_Procedure
```

Visual Objects

```
METHOD OCX_MouseDown(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_MouseDown(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic

```
function MouseDown as v (Button as N,Shift as N,X as
OLE::Exontrol.Tree.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Tree.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_MouseDown(Button,Shift,X,Y)
return
```

The following VB sample prints the cell's caption that has been clicked:

```
Private Sub Tree1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
    ' 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 EXTREELibCtl.HitTestInfoEnum
    ' Gets the item from (X,Y)
    h = Tree1.ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Then
        Debug.Print Tree1.Items.CellCaption(h, c) & " HT = " & hit
    End If
End Sub
```

If you need to add a context menu based on the item you can use the MouseUp event, like in the following VB sample (the sample uses the Exontrol's ExPopupMenu Component):

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If (Button = 2) Then
        ' Converts the container coordinates to client coordinates
        X = X / Screen.TwipsPerPixelX
        Y = Y / Screen.TwipsPerPixelY
        Dim h As HITEM
        Dim c As Long, hit as Long
        ' Gets the item from (X,Y)
        h = Tree1.ItemFromPoint(X, Y, c, hit)
        If Not (h = 0) Then
            Dim i As Long
            PopupMenu1.Items.Add Tree1.Items.CellCaption(h, c)
            i = PopupMenu1.ShowAtCursor
        End If
    End If
End Sub
```

The following VC sample displays the caption of the cell being clicked:

```
#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 OnMouseDownTree1(short Button, short Shift, long X, long Y)
{
    int c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
    if ( ( hItem != 0 ) || ( c != 0 ) )
    {
        CItems items = m_tree.GetItems();
        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 displays the caption from the cell being clicked:

```
Private Sub AxTree1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseDownEvent) Handles AxTree1.MouseDownEvent
    With AxTree1
        Dim i As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (Not (i = 0) Or Not (c = 0)) Then
            Debug.WriteLine("Cell: " & .Items.CellCaption(i, c) & " Hit: " & hit.ToString())
        End If
    End With
```

```
End Sub
```

The following C# sample displays the caption from the cell being clicked:

```
private void axTree1_MouseDownEvent(object sender,
AxEXTREELib._ITreeEvents_MouseDownEvent e)
{
   int c = 0;
   EXTREELib.HitTestInfoEnum hit;
   int i = axTree1.get_ItemFromPoint( e.x, e.y, out c,out hit );
   if ( ( i != 0 ) || ( c != 0 ) )
   {
      string s = axTree1.Items.get_CellCaption( i,c ).ToString();
      s = "Cell: " + s + ", Hit: " + hit.ToString();
      System.Diagnostics.Debug.WriteLine( s );
   }
}
```

The following VFP sample displays the caption from the cell being clicked:

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

local c, hit
c = 0
hit = 0
with thisform.Tree1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
      wait window nowait .Items.CellCaption( 0, c ) + " " + Str( hit )
   endif
endwith
```

event MouseMove (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user moves the mouse.

Type	Description
Button as Integer	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. Gets which mouse button was pressed as 1 for Left Mouse Button, 2 for Right Mouse Button and 4 for Middle Mouse Button.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders. Use the Background(exCursorHoverColumn) property to specify the visual appearance of the column when the cursor hovers the column. Use the ItemFromPoint property to get the item from cursor. Use the ColumnFromPoint property to get the column from point:

Syntax for MouseMove event, **/NET** version, on:

C#
```
private void MouseMoveEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub MouseMoveEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseMoveEvent
End Sub
```

Syntax for MouseMove event, **/COM** version, on:

C#

```
private void MouseMoveEvent(object sender,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_MouseMoveEvent);
begin
end;
```

Powe…

```
begin event MouseMove(integer Button,integer Shift,long X,long Y)
end event MouseMove
```

VB.NET

```
Private Sub MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_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(oTree,Button,Shift,X,Y)
RETURN
```

Syntax for MouseMove event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="MouseMove(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function MouseMove(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComMouseMove Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseMove llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_MouseMove(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_MouseMove(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function MouseMove as v (Button as N,Shift as N,X as
OLE::Exontrol.Tree.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Tree.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_MouseMove(Button,Shift,X,Y)
return
```

The following VB sample prints the cell's caption from the cursor (if the control contains no

inner cells. Use the SplitCell property to insert inner cells) :

```
Private Sub Tree1_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 EXTREELibCtl.HitTestInfoEnum
   ' Gets the item from (X,Y)
   h = Tree1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Then
      Debug.Print Tree1.Items.CellCaption(h, c) & " HT = " & hit
   End If
End Sub
```

The following VB sample displays the cell's caption from the cursor (if the control contains inner cells):

```
Private Sub Tree1_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 EXTREELibCtl.HitTestInfoEnum
   ' Gets the item from (X,Y)
   h = Tree1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Or Not (c = 0) Then
      Debug.Print Tree1.Items.CellCaption(h, c) & " HT = " & hit
   End If
End Sub
```

The following C++ sample displays the cell's from the point:

```
#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 OnMouseMoveTree1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
   if ( ( hItem != 0 ) || ( c != 0 ) )
   {
      CItems items = m_tree.GetItems();
      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 displays the cell's from the point:

```
Private Sub AxTree1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseMoveEvent) Handles AxTree1.MouseMoveEvent
   With AxTree1
      Dim i As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
      i = .get_ItemFromPoint(e.x, e.y, c, hit)
      If (Not (i = 0) Or Not (c = 0)) Then
         Debug.WriteLine("Cell: " & .Items.CellCaption(i, c) & " Hit: " & hit.ToString())
```

```
    End If
  End With
End Sub
```

The following C# sample displays the cell's from the point:

```
private void axTree1_MouseMoveEvent(object sender,
AxEXTREELib._ITreeEvents_MouseMoveEvent e)
{
  int c = 0;
  EXTREELib.HitTestInfoEnum hit;
  int i = axTree1.get_ItemFromPoint( e.x, e.y, out c,out hit );
  if ( ( i != 0 ) || ( c != 0 ) )
  {
    object cap = axTree1.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 displays the cell's from the point:

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

local c, hit
c = 0
hit = 0
with thisform.Tree1
  .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
  if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
    wait window nowait .Items.CellCaption( 0, c ) + " " + Str( hit )
  endif
endwith
```

event MouseUp (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user releases a mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the ItemFromPoint property to get the item from point. Use the ColumnFromPoint property to get the column from point.

Syntax for MouseUp event, **/NET** version, on:

C#

```
private void MouseUpEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB

```
Private Sub MouseUpEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseUpEvent
End Sub
```

Syntax for MouseUp event, **/COM** version, on:

C#

```
private void MouseUpEvent(object sender,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_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(oTree,Button,Shift,X,Y)
RETURN
```

Syntax for MouseUp event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="MouseUp(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function MouseUp(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComMouseUp Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseUp llButton llShift llX llY
End_Procedure
```

Visual Objects

```
METHOD OCX_MouseUp(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_MouseUp(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic

```
function MouseUp as v (Button as N,Shift as N,X as
OLE::Exontrol.Tree.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Tree.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_MouseUp(Button,Shift,X,Y)
return
```

The following VB sample prints the cell's caption where the mouse has been released:

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
    ' Converts the container coordinates to client coordinates
    X = X / Screen.TwipsPerPixelX
    Y = Y / Screen.TwipsPerPixelY
    Dim h As HITEM
    Dim c As Long, hit as Long
    ' Gets the item from (X,Y)
    h = Tree1.ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Then
        Debug.Print Tree1.Items.CellCaption(h, c)
    End If
End Sub
```

If you need to add a context menu based on the item you can use the MouseUp event, like in the following VB sample (the sample uses the Exontrol's ExPopupMenu Component):

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If (Button = 2) Then
        ' Converts the container coordinates to client coordinates
        X = X / Screen.TwipsPerPixelX
        Y = Y / Screen.TwipsPerPixelY
        Dim h As HITEM
        Dim c As Long, hit as Long
        ' Gets the item from (X,Y)
        h = Tree1.ItemFromPoint(X, Y, c, hit)
        If Not (h = 0) Then
            Dim i As Long
            PopupMenu1.Items.Add Tree1.Items.CellCaption(h, c)
            i = PopupMenu1.ShowAtCursor
        End If
    End If
End Sub
```

The following VC sample displays the caption of the cell where the mouse is released:

```
#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 OnMouseUpTree1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
    if ( ( hItem != 0 ) || ( c != 0 ) )
    {
        CItems items = m_tree.GetItems();
        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 displays the caption of the cell where the mouse is released:

```
Private Sub AxTree1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseUpEvent) Handles AxTree1.MouseUpEvent
    With AxTree1
        Dim i As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (Not (i = 0) Or Not (c = 0)) Then
            Debug.WriteLine("Cell: " & .Items.CellCaption(i, c) & " Hit: " & hit.ToString())
        End If
    End With
End Sub
```

The following C# sample displays the caption of the cell where the mouse is released:

```
private void axTree1_MouseUpEvent(object sender,
AxEXTREELib._ITreeEvents_MouseUpEvent e)
{
   int c = 0;
   EXTREELib.HitTestInfoEnum hit;
   int i = axTree1.get_ItemFromPoint( e.x, e.y, out c,out hit );
   if ( ( i != 0 ) || ( c != 0 ) )
   {
      string s = axTree1.Items.get_CellCaption( i,c ).ToString();
      s = "Cell: " + s + ", Hit: " + hit.ToString();
      System.Diagnostics.Debug.WriteLine( s );
   }
}
```

The following VFP sample displays the caption of the cell where the mouse is released:

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

local c, hit
c = 0
hit = 0
with thisform.Tree1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
      wait window nowait .Items.CellCaption( 0, c ) + " " + Str( hit )
   endif
endwith
```

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 scroll bar has changed.
NewVal as Long	A long value that indicates the new scroll bar value in pixels.

If the control has no scroll bars the OffsetChanged and OversizeChanged events are not fired. Use the ScrollBars property of the control to determine which scroll bars are visible within the control. Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the ScrollToolTip property to specify the tooltip being displayed when the user clicks and moves the scrollbar's thumb. Use the Background property to change the visual appearance for any part in the control's scroll bar.

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,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_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(oTree,Horizontal,NewVal)
RETURN
```

Syntax for OffsetChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OffsetChanged(Horizontal,NewVal)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OffsetChanged(Horizontal,NewVal)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOffsetChanged Boolean llHorizontal Integer llNewVal
    Forward Send OnComOffsetChanged llHorizontal llNewVal
End_Procedure
```

Visual Objects

```
METHOD OCX_OffsetChanged(Horizontal,NewVal) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OffsetChanged(boolean _Horizontal,int _NewVal)
{
}
```

XBasic

```
function OffsetChanged as v (Horizontal as L,NewVal as N)
end function
```

dBASE

```
function nativeObject_OffsetChanged(Horizontal,NewVal)
return
```

The following VB sample displays the new scroll position when user scrolls horizontally the control:

```
Private Sub Tree1_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 OnOffsetChangedTree1(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 AxTree1_OffsetChanged(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_OffsetChangedEvent) Handles AxTree1.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 axTree1_OffsetChanged(object sender,
AxEXTREELib._ITreeEvents_OffsetChangedEvent e)
{
    if ( !e.horizontal )
        System.Diagnostics.Debug.WriteLine(e.newVal);
}
```

The following VFP sample displays the new scroll position when the user scrolls vertically the control:

```
*** ActiveX Control Event ***
LPARAMETERS horizontal, newval

if ( 0 # horizontal )
    wait window nowait str( newval )
endif
```

event OLECompleteDrag (Effect as Long)

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled

Type	Description
Effect as Long	A long set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another

The OLECompleteDrag event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the effect parameter of the OLEDragDrop event. Based on this, the source can then determine the appropriate action it needs to take. For example, if the object was moved into the target (exDropEffectMove), the source needs to delete the object from itself after the move. The control supports only manual OLE drag and drop events. In order to enable OLE drag and drop feature into control you have to set the OLEDropMode and OLEDrag properties.

The settings for Effect are:

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

Syntax for OLECompleteDrag event, **/NET** version, on:

C#
```
// OLECompleteDrag event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLECompleteDrag event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLECompleteDrag event, **/COM** version, on:

C#
```
private void OLECompleteDrag(object sender,
AxEXTREELib._ITreeEvents_OLECompleteDragEvent e)
{
```

```
}
```

C++

```
void OnOLECompleteDrag(long Effect)
{
}
```

C++ Builder

```
void __fastcall OLECompleteDrag(TObject *Sender,long Effect)
{
}
```

Delphi

```
procedure OLECompleteDrag(ASender: TObject; Effect : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OLECompleteDrag(sender: System.Object; e:
AxEXTREELib._ITreeEvents_OLECompleteDragEvent);
begin
end;
```

Powe…

```
begin event OLECompleteDrag(long Effect)
end event OLECompleteDrag
```

VB.NET

```
Private Sub OLECompleteDrag(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OLECompleteDragEvent) Handles OLECompleteDrag
End Sub
```

VB6

```
Private Sub OLECompleteDrag(ByVal Effect As Long)
End Sub
```

VBA

```
Private Sub OLECompleteDrag(ByVal Effect As Long)
End Sub
```

VFP

```
LPARAMETERS Effect

```

Xbas…

```
PROCEDURE OnOLECompleteDrag(oTree,Effect)
RETURN
```

Syntax for OLECompleteDrag event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLECompleteDrag(Effect)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLECompleteDrag(Effect)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLECompleteDrag Integer llEffect
    Forward Send OnComOLECompleteDrag llEffect
End_Procedure
```

Visual Objects

```
METHOD OCX_OLECompleteDrag(Effect) CLASS MainDialog
RETURN NIL
```

X++

```
// OLECompleteDrag event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

XBasic

```
function OLECompleteDrag as v (Effect as N)
end function
```

dBASE

```
function nativeObject_OLECompleteDrag(Effect)
return
```

event OLEDragDrop (Data as ExDataObject, Effect as Long, Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here.
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Remarks.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
Shift as Integer	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

In the /NET Assembly, you have to use the DragDrop event as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

The OLEDragDrop event is fired when the user has dropped files or clipboard information into the control. Use the OLEDropMode property on exOLEDropManual to enable OLE drop and drop support. Use the ItemFromPoint property to get the item from point. Use the ColumnFromPoint property to get the column from point. Use the AddItem method to add a new item to the control. Use the InsertItem method to insert a new child item. Use the ItemPosition property to specify the item's position.

The settings for Effect are:

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

Syntax for OLEDragDrop event, **/NET** version, on:

C#
```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLEDragDrop event, **/COM** version, on:

C#
```
private void OLEDragDrop(object sender, AxEXTREELib._ITreeEvents_OLEDragDropEvent e)
{
}
```

C++
```
void OnOLEDragDrop(LPDISPATCH Data,long FAR* Effect,short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall OLEDragDrop(TObject *Sender,Extreelib_tlb::IExDataObject *Data,long *
Effect,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure OLEDragDrop(ASender: TObject; Data : IExDataObject;var Effect :
Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OLEDragDrop(sender: System.Object; e:
AxEXTREELib._ITreeEvents_OLEDragDropEvent);
begin
end;
```

Powe…

```
begin event OLEDragDrop(oleobject Data,long Effect,integer Button,integer
Shift,long X,long Y)
end event OLEDragDrop
```

VB.NET

```
Private Sub OLEDragDrop(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OLEDragDropEvent) Handles OLEDragDrop
End Sub
```

VB6

```
Private Sub OLEDragDrop(ByVal Data As EXTREELibCtl.IExDataObject,Effect As
Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As
Single)
End Sub
```

VBA

```
Private Sub OLEDragDrop(ByVal Data As Object,Effect As Long,ByVal Button As
Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Data,Effect,Button,Shift,X,Y
```

Xbas…

```
PROCEDURE OnOLEDragDrop(oTree,Data,Effect,Button,Shift,X,Y)
```

```
RETURN
```

Syntax for OLEDragDrop event, **/COM** version (others), on:

Java...
```
<SCRIPT EVENT="OLEDragDrop(Data,Effect,Button,Shift,X,Y)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc...
```
<SCRIPT LANGUAGE="VBScript">
Function OLEDragDrop(Data,Effect,Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data...
```
Procedure OnComOLEDragDrop Variant llData Integer llEffect Short llButton
Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY
    Forward Send OnComOLEDragDrop llData llEffect llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_OLEDragDrop(Data,Effect,Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
// OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

XBasic
```
function OLEDragDrop as v (Data as OLE::Exontrol.Tree.1::IExDataObject,Effect as
N,Button as N,Shift as N,X as OLE::Exontrol.Tree.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Tree.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_OLEDragDrop(Data,Effect,Button,Shift,X,Y)
return
```

The following VB sample adds a new item when the user drags a file (Open the Windows Explorer, click and drag a file to the control) :

```
Private Sub Tree1_OLEDragDrop(Index As Integer, ByVal Data As
EXTREELibCtl.IExDataObject, Effect As Long, ByVal Button As Integer, ByVal Shift As Integer,
```

```
ByVal X As Single, ByVal Y As Single)
   If Data.GetFormat(exCFFiles) Then
      Data.GetData (exCFFiles)
      Dim strFile As String
      strFile = Data.Files(0)
      'Adds a new item to the control
      Tree1(Index).Visible = False
      With Tree1(Index)
      .BeginUpdate
         Dim i As HITEM
         i = .Items.AddItem(strFile)
         .Items.EnsureVisibleItem i
      .EndUpdate
      End With
      Tree1(Index).Visible = True
   End If
End Sub
```

The following VC sample inserts a child item for each file that user drags:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

#include "Items.h"
void OnOLEDragDropTree1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
   EXTREELib::IExDataObjectPtr spData( Data );
   if ( spData != NULL )
      if ( spData->GetFormat( EXTREELib::exCFFiles ) )
      {
         CItems items = m_tree.GetItems();
         // Gets the handle of the item where the files will be inserted
         long c = 0, h = 0, nParentItem = m_tree.GetItemFromPoint( X, Y, &c, &h );
         if ( nParentItem == 0 )
            if ( c != 0 )
               nParentItem = items.GetCellItem( c );
         EXTREELib::IExDataObjectFilesPtr spFiles( spData->Files );
```

```
        if ( spFiles->Count > 0 )
        {
            m_tree.BeginUpdate();
            COleVariant vtMissing; vtMissing.vt = VT_ERROR;
            for ( long i = 0; i < spFiles->Count; i++ )
                items.InsertItem( nParentItem, vtMissing, COleVariant( spFiles->GetItem( i
).operator const char *() ) );
            if ( nParentItem )
                items.SetExpandItem( nParentItem, TRUE );
            m_tree.EndUpdate();
        }

    }
}
```

The #import statement imports definition for the ExDataObject and ExDataObjectFiles objects. If the extree.dll file is located in another folder than the system folder, the path to the file must be specified. The sample gets the item where the files were dragged and insert all files in that position, as child items, if case.

The following VB.NET sample inserts a child item for each file that user drags:

```
Private Sub AxTree1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_OLEDragDropEvent) Handles AxTree1.OLEDragDrop
    If e.data.GetFormat(EXTREELib.exClipboardFormatEnum.exCFFiles) Then
        If (e.data.Files.Count > 0) Then
            AxTree1.BeginUpdate()
            With AxTree1.Items
                Dim iParent As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
                iParent = AxTree1.get_ItemFromPoint(e.x, e.y, c, hit)
                If iParent = 0 Then
                    If Not c = 0 Then
                        iParent = .CellItem(c)
                    End If
                End If
                Dim i As Long
                For i = 0 To e.data.Files.Count - 1
                    .InsertItem(iParent, , e.data.Files(i))
```

```
            Next
            If Not (iParent = 0) Then
               .ExpandItem(iParent) = True
            End If
         End With
         AxTree1.EndUpdate()
      End If
   End If
End Sub
```

The following C# sample inserts a child item for each file that user drags:

```
private void axTree1_OLEDragDrop(object sender,
AxEXTREELib._ITreeEvents_OLEDragDropEvent e)
{
   if ( e.data.GetFormat( Convert.ToInt16(EXTREELib.exClipboardFormatEnum.exCFFiles) ) )
      if ( e.data.Files.Count > 0 )
      {
         EXTREELib.HitTestInfoEnum hit;
         int c = 0, iParent = axTree1.get_ItemFromPoint( e.x, e.y, out c, out hit );
         if ( iParent == 0 )
            if ( c != 0 )
               iParent = axTree1.Items.get_CellItem( c );

         axTree1.BeginUpdate();
         for ( int i = 0; i < e.data.Files.Count; i++ )
            axTree1.Items.InsertItem( iParent,"", e.data.Files[i].ToString() );
         if ( iParent != 0 )
            axTree1.Items.set_ExpandItem( iParent, true );
         axTree1.EndUpdate();
      }
}
```

The following VFP sample inserts a child item for each file that user drags:

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

```
local c, hit, iParent
c = 0
hit = 0
if ( data.GetFormat( 15 ) ) && exCFFiles
   if ( data.Files.Count() > 0 )
      with thisform.Tree1.Items
         iParent = thisform.Tree1.ItemFromPoint( x, y, @c, @hit )

         thisform.Tree1.BeginUpdate()
         for i = 0 to data.files.Count() - 1
            .InsertItem( iParent, "", data.files(i) )
         next
         if ( iParent != 0 )
            .DefaultItem = iParent
            .ExpandItem( 0 ) = .t.
         endif
         thisform.Tree1.EndUpdate()
      endwith
   endif
endif
```

event OLEDragOver (Data as ExDataObject, Effect as Long, Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, State as Integer)

Occurs when one component is dragged over another.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Remarks.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
Shift as Integer	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

State as Integer	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed in Remarks.

The settings for effect are:

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

The settings for state are:

- exOLEDragEnter (0), Source component is being dragged within the range of a target.
- exOLEDragLeave (1), Source component is being dragged out of the range of a target.
- exOLEOLEDragOver (2), Source component has moved from one position in the target to another.

Note If the state parameter is 1, indicating that the mouse pointer has left the target, then the x and y parameters will contain zeros.
The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.
For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:

If Effect = exOLEDropEffectCopy...
Instead, the source component should mask for the value or values being sought, such as this:
If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...
-or-
If (Effect And exOLEDropEffectCopy)...
This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.
The control supports only manual OLE drag and drop events.

Syntax for OLEDragOver event, **/NET** version, on:

C#
```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB

```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

Syntax for OLEDragOver event, **/COM** version, on:

C#

```
private void OLEDragOver(object sender,
AxEXTREELib._ITreeEvents_OLEDragOverEvent e)
{
}
```

C++

```
void OnOLEDragOver(LPDISPATCH Data,long FAR* Effect,short Button,short
Shift,long X,long Y,short State)
{
}
```

C++ Builder

```
void __fastcall OLEDragOver(TObject *Sender,Extreelib_tlb::IExDataObject
*Data,long * Effect,short Button,short Shift,int X,int Y,short State)
{
}
```

Delphi

```
procedure OLEDragOver(ASender: TObject; Data : IExDataObject;var Effect :
Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer;State : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OLEDragOver(sender: System.Object; e:
AxEXTREELib._ITreeEvents_OLEDragOverEvent);
begin
end;
```

Powe…

```
begin event OLEDragOver(oleobject Data,long Effect,integer Button,integer
Shift,long X,long Y,integer State)
end event OLEDragOver
```

VB.NET

```
Private Sub OLEDragOver(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

VB6

```
Private Sub OLEDragOver(ByVal Data As EXTREELibCtl.IExDataObject,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As Single,ByVal State As Integer)
End Sub
```

VBA

```
Private Sub OLEDragOver(ByVal Data As Object,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long,ByVal State As Integer)
End Sub
```

VFP

```
LPARAMETERS Data,Effect,Button,Shift,X,Y,State

```

Xbas…

```
PROCEDURE OnOLEDragOver(oTree,Data,Effect,Button,Shift,X,Y,State)
RETURN
```

Syntax for OLEDragOver event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLEDragOver(Data,Effect,Button,Shift,X,Y,State)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLEDragOver Variant llData Integer llEffect Short llButton Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY Short llState
    Forward Send OnComOLEDragOver llData llEffect llButton llShift llX llY llState
End_Procedure
```

Visual Objects

```
METHOD OCX_OLEDragOver(Data,Effect,Button,Shift,X,Y,State) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,

```

DragDrop ... events.

XBasic

```
function OLEDragOver as v (Data as OLE::Exontrol.Tree.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.Tree.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Tree.1::OLE_YPOS_PIXELS,State as N)
end function
```

dBASE

```
function nativeObject_OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
return
```

event OLEGiveFeedback (Effect as Long, DefaultCursors as Boolean)

Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.

Type	Description
Effect as Long	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed in Remarks.
DefaultCursors as Boolean	Boolean value that determines whether to use the default mouse cursor, or to use a user-defined mouse cursor.True (default) = use default mouse cursor.False = do not use default cursor. Mouse cursor must be set with the MousePointer property of the Screen object.

The settings for Effect are:

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

If there is no code in the OLEGiveFeedback event, or if the defaultcursors parameter is set to True, the mouse cursor will be set to the default cursor provided by the control. The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:
If Effect = exOLEDropEffectCopy...
Instead, the source component should mask for the value or values being sought, such as this:
If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...
-or-
If (Effect And exOLEDropEffectCopy)...
This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

The control supports only manual OLE drag and drop events.

Syntax for OLEGiveFeedback event, **/NET** version, on:

C#
```
// OLEGiveFeedback event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLEGiveFeedback event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLEGiveFeedback event, **/COM** version, on:

C#
```
private void OLEGiveFeedback(object sender,
AxEXTREELib._ITreeEvents_OLEGiveFeedbackEvent e)
{
}
```

C++
```
void OnOLEGiveFeedback(long Effect,BOOL FAR* DefaultCursors)
{
}
```

C++ Builder
```
void __fastcall OLEGiveFeedback(TObject *Sender,long Effect,VARIANT_BOOL *
DefaultCursors)
{
}
```

Delphi
```
procedure OLEGiveFeedback(ASender: TObject; Effect : Integer;var DefaultCursors
: WordBool);
begin
end;
```

Delphi 8 (.NET only)
```
procedure OLEGiveFeedback(sender: System.Object; e:
AxEXTREELib._ITreeEvents_OLEGiveFeedbackEvent);
begin
end;
```

Powe…
```
begin event OLEGiveFeedback(long Effect,boolean DefaultCursors)
end event OLEGiveFeedback
```

VB.NET

```
Private Sub OLEGiveFeedback(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OLEGiveFeedbackEvent) Handles OLEGiveFeedback
End Sub
```

VB6

```
Private Sub OLEGiveFeedback(ByVal Effect As Long,DefaultCursors As Boolean)
End Sub
```

VBA

```
Private Sub OLEGiveFeedback(ByVal Effect As Long,DefaultCursors As Boolean)
End Sub
```

VFP

```
LPARAMETERS Effect,DefaultCursors
```

Xbas…

```
PROCEDURE OnOLEGiveFeedback(oTree,Effect,DefaultCursors)
RETURN
```

Syntax for OLEGiveFeedback event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLEGiveFeedback(Effect,DefaultCursors)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLEGiveFeedback(Effect,DefaultCursors)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLEGiveFeedback Integer llEffect Boolean llDefaultCursors
    Forward Send OnComOLEGiveFeedback llEffect llDefaultCursors
End_Procedure
```

Visual Objects

```
METHOD OCX_OLEGiveFeedback(Effect,DefaultCursors) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEGiveFeedback event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

XBasic

```
function OLEGiveFeedback as v (Effect as N,DefaultCursors as L)
end function
```

dBASE

```
function nativeObject_OLEGiveFeedback(Effect,DefaultCursors)
return
```

event OLESetData (Data as ExDataObject, Format as Integer)

Occurs on a drag source when a drop target calls the GetData method and there is no data in a specified format in the OLE drag-and-drop DataObject.

Type	Description
Data as ExDataObject	An ExDataObject object in which to place the requested data. The component calls the SetData method to load the requested format.
Format as Integer	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the ExDataObject object.

The OLESetData is not currently supported.

Syntax for OLESetData event, **/NET** version, on:

C#
```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLESetData event, **/COM** version, on:

C#
```
private void OLESetData(object sender,
AxEXTREELib._ITreeEvents_OLESetDataEvent e)
{
}
```

C++
```
void OnOLESetData(LPDISPATCH Data,short Format)
{
}
```

C++ Builder
```
void __fastcall OLESetData(TObject *Sender,Extreelib_tlb::IExDataObject
*Data,short Format)
{
}
```

Delphi

```
procedure OLESetData(ASender: TObject; Data : IExDataObject;Format : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure OLESetData(sender: System.Object; e:
AxEXTREELib._ITreeEvents_OLESetDataEvent);
begin
end;
```

Powe…

```
begin event OLESetData(oleobject Data,integer Format)
end event OLESetData
```

VB.NET

```
Private Sub OLESetData(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OLESetDataEvent) Handles OLESetData
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXTREELibCtl.IExDataObject,ByVal Format
As Integer)
End Sub
```

VBA

```
Private Sub OLESetData(ByVal Data As Object,ByVal Format As Integer)
End Sub
```

VFP

```
LPARAMETERS Data,Format

```

Xbas…

```
PROCEDURE OnOLESetData(oTree,Data,Format)
RETURN
```

Syntax for OLESetData event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLESetData(Data,Format)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLESetData(Data,Format)
End Function
```

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLESetData Variant llData Short llFormat
    Forward Send OnComOLESetData llData llFormat
End_Procedure
```

Visual Objects

```
METHOD OCX_OLESetData(Data,Format) CLASS MainDialog
RETURN NIL
```

X++

```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

XBasic

```
function OLESetData as v (Data as OLE::Exontrol.Tree.1::IExDataObject,Format as
N)
end function
```

dBASE

```
function nativeObject_OLESetData(Data,Format)
return
```

event OLEStartDrag (Data as ExDataObject, AllowedEffects as Long)

Occurs when the OLEDrag method is called.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The programmer should provide the values for this parameter in this event. The SetData and Clear methods cannot be used here.
AllowedEffects as Long	A long containing the effects that the source component supports. The possible values are listed in Settings. The programmer should provide the values for this parameter in this event

In the /NET Assembly, you have to use the DragEnter event as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

Use the Background(exDragDropBefore) property to specify the visual appearance for the dragging items, before painting the items. Use the Background(exDragDropAfter) property to specify the visual appearance for the dragging items, after painting the items. Use the Background(exDragDropList) property to specify the graphic feedback for the item from the cursor, while the OLE drag and drop operation is running.

The settings for AllowEffects are:

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

The source component should logically Or together the supported values and places the result in the AllowedEffects parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be). You may wish to defer putting data into the ExDataObject object until the target component requests it. This allows the source component to save time. If the user does not load any formats into the ExDataObject, then the drag/drop operation is canceled. Use exCFFiles and Files property to add files to the drag and drop data object.

Syntax for OLEStartDrag event, **/NET** version, on:

C#
```
// OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLEStartDrag event, **/COM** version, on:

C#
```
private void OLEStartDrag(object sender,
AxEXTREELib._ITreeEvents_OLEStartDragEvent e)
{
}
```

C++
```
void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
{
}
```

C++ Builder
```
void __fastcall OLEStartDrag(TObject *Sender,Extreelib_tlb::IExDataObject
*Data,long * AllowedEffects)
{
}
```

Delphi
```
procedure OLEStartDrag(ASender: TObject; Data : IExDataObject;var
AllowedEffects : Integer);
begin
end;
```

Delphi 8 (.NET only)
```
procedure OLEStartDrag(sender: System.Object; e:
AxEXTREELib._ITreeEvents_OLEStartDragEvent);
begin
end;
```

Powe…
```
begin event OLEStartDrag(oleobject Data,long AllowedEffects)
end event OLEStartDrag
```

VB.NET
```
Private Sub OLEStartDrag(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_OLEStartDragEvent) Handles OLEStartDrag
```

```
End Sub
```

VB6

```
Private Sub OLEStartDrag(ByVal Data As
EXTREELibCtl.IExDataObject,AllowedEffects As Long)
End Sub
```

VBA

```
Private Sub OLEStartDrag(ByVal Data As Object,AllowedEffects As Long)
End Sub
```

VFP

```
LPARAMETERS Data,AllowedEffects

```

Xbas…

```
PROCEDURE OnOLEStartDrag(oTree,Data,AllowedEffects)
RETURN
```

Syntax for OLEStartDrag event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLEStartDrag(Data,AllowedEffects)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLEStartDrag(Data,AllowedEffects)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLEStartDrag Variant llData Integer llAllowedEffects
    Forward Send OnComOLEStartDrag llData llAllowedEffects
End_Procedure
```

Visual Objects

```
METHOD OCX_OLEStartDrag(Data,AllowedEffects) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

XBasic

```
function OLEStartDrag as v (Data as
OLE::Exontrol.Tree.1::IExDataObject,AllowedEffects as N)
end function
```

```
dBASE
function nativeObject_OLEStartDrag(Data,AllowedEffects)
return
```

The idea of drag and drop in exTree control is the same as in other controls. To start accepting drag and drop sources the exTree control should have the OLEDropMode to exOLEDropManual. Once that is is set, the exTree starts accepting any drag and drop sources.

The first step is if you want to be able to drag items from your exTree control to other controls the idea is to handle the OLE_StartDrag event. The event passes an object ExDataObject (Data) as argument. The Data and AllowedEffects can be changed only in the OLEStartDrag event. The OLE_StartDrag event is fired when user is about to drag items from the control. **The AllowedEffect parameter and SetData property must be set to continue drag and drop operation, as in the following samples**:

The following VB sample drags data from a control to another, by registering a new clipboard format:

```
Private Sub Tree1_OLEStartDrag(Index As Integer, ByVal Data As
EXTREELibCtl.IExDataObject, AllowedEffects As Long)

  ' We are going to add two clipboard formats: text and "EXTREE" clipboard format.
  ' We need to use RegisterClipboardFormat API function in order to register our
  ' clipboard format. One cliboard format is enough, but the sample shows
  ' how to filter in OLEDragDrop event the other clipboard formats

  ' Builds a string that contains each cell's caption on a new line
  Dim n As Long
  Dim s As String
  With Tree1(Index)
    s = Index & vbCrLf ' Saves the source
    For n = 0 To .Columns.Count - 1
      s = s & .Items.CellCaption(.Items.SelectedItem(0), n) & vbCrLf
    Next
  End With

  AllowedEffects = 0
  ' Checks whether the selected item has a parent
```

```
    If (Tree1(Index).Items.ItemParent(Tree1(Index).Items.SelectedItem(0)) <> 0) Then
        AllowedEffects = 1
    End If
    ' Sets the text clipboard format
    Data.SetData s, exCFText

    ' Builds an array of bytes, and copy there all characters in the s string.
    ' Passes the array to the SetData method.
    ReDim v(Len(s)) As Byte
    For n = 0 To Len(s) - 1
        v(n) = Asc(Mid(s, n + 1, 1))
    Next
    Data.SetData v, RegisterClipboardFormat("EXTREE")

End Sub
```

The code fills data for two types of clipboard formats: text (CF_TEXT) and "EXTREE" registered clipboard format. The registered clipboard format must be an array of bytes. As you can see we have used the RegisterClipboardFormat API function, and it should be declared like:

```
Private Declare Function RegisterClipboardFormat Lib "user32" Alias
"RegisterClipboardFormatA" (ByVal lpString As String) As Integer
```

The second step is accepting OLE drag and drop source objects. That means, if you would like to let your control accept drag and drop objects, you have to handle the OLEDragDrop event. It gets as argument an object Data that stores the drag and drop information. The next sample shows how handle the OLEDragDrop event:

```
Private Sub Tree1_OLEDragDrop(Index As Integer, ByVal Data As
EXTREELibCtl.IExDataObject, Effect As Long, ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
    ' Checks whether the clipboard format is our. Since we have registered the clipboard in
the
    ' OLEStartData format we now its format, so we can handle this type of clip formats.
    If (Data.GetFormat(RegisterClipboardFormat("EXTREE"))) Then
        ' Builds the saved string from the array passed
        Dim s As String
        Dim v() As Byte
```

```
    Dim n As Integer
    v = Data.GetData(RegisterClipboardFormat("EXTREE"))
    For n = LBound(v) To UBound(v)
       s = s + Chr(v(n))
    Next
    Debug.Print s

    'Adds a new item to the control, and sets the cells captions like we saved, line by line
    Tree1(Index).Visible = False
    With Tree1(Index)
       .BeginUpdate
       Dim i As HITEM
       Dim item As String
       Dim nCur As Long
       i = .Items.AddItem()
       nCur = InStr(1, s, vbCrLf) + Len(vbCrLf) ' Jumps the source
       For n = 0 To .Columns.Count - 1
          Dim nnCur As Long
          nnCur = InStr(nCur, s, vbCrLf)
          .Items.CellCaption(i, n) = Mid(s, nCur, nnCur - nCur)
          nCur = nnCur + Len(vbCrLf)
       Next
       .Items.CellImage(i, "EmployeeID") = Int(.Items.CellCaption(i, "EmployeeID"))
       .Items.SetParent i, h(Index, Int(.Items.CellCaption(i, "EmployeeID")) - 1)
       .Items.EnsureVisibleItem i
       .EndUpdate
    End With
    Tree1(Index).Visible = True
  End If
End Sub
```

The following VC sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
#import <extree.dll> rename( "GetItems", "exGetItems" )

#include "Items.h"
#include "Columns.h"
```

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}

void OnOLEStartDragTree1(LPDISPATCH Data, long FAR* AllowedEffects)
{
   CItems items = m_tree.GetItems();
   long nCount = items.GetSelectCount(), nColumnCount =
m_tree.GetColumns().GetCount();
   if ( nCount > 0 )
   {
      *AllowedEffects = /*exOLEDropEffectCopy */ 1;
      EXTREELib::IExDataObjectPtr spData( Data );
      if ( spData !=NULL )
      {
         CString strData;
         for ( long i = 0; i < nCount; i++ )
         {
            COleVariant vtItem( items.GetSelectedItem( i ) );
            for ( long j = 0; j < nColumnCount; j++ )
               strData += V2S( &items.GetCellCaption( vtItem, COleVariant( j ) ) ) + "\t";
         }
         strData += "\r\n";
         spData->SetData( COleVariant( strData ), COleVariant( (long)EXTREELib::exCFText) );
      }
   }
```

```
}
```

The sample saves data as CF_TEXT format (EXTREELib::exCFText). The data is a text, where each item is separated by "\r\n" (new line), and each cell is separated by "\t" (TAB charcater). Of course, data can be saved as you want. The sample only gives an idea of what and how it could be done. The sample uses the #import statement to import the control's type library, including definitions for ExDataObject and ExDataObjectFiles that are required to fill data to be dragged. If your extree.dll file is located in another place than your system folder, the path to the extree.dll file needs to be specified, else compiler errors occur.

The following VB.NET sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
Private Sub AxTree1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_OLEStartDragEvent) Handles AxTree1.OLEStartDrag
   With AxTree1.Items
      If (.SelectCount > 0) Then
         e.allowedEffects = 1 'exOLEDropEffectCopy
         Dim i As Integer, j As Integer, strData As String, nColumnCount As Long =
AxTree1.Columns.Count
         For i = 0 To .SelectCount - 1
            For j = 0 To nColumnCount - 1
               strData = strData + .CellCaption(.SelectedItem(i), j) + Chr(Keys.Tab)
            Next
         Next
         strData = strData + vbCrLf
         e.data.SetData(strData, EXTREELib.exClipboardFormatEnum.exCFText)
      End If
   End With
End Sub
```

The following C# sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
private void axTree1_OLEStartDrag(object sender,
AxEXTREELib._ITreeEvents_OLEStartDragEvent e)
{
   int nCount = axTree1.Items.SelectCount;
   if ( nCount > 0 )
```

```
    {
        int nColumnCount = axTree1.Columns.Count;
        e.allowedEffects = /*exOLEDropEffectCopy*/ 1;
        string strData = "";
        for ( int i =0 ; i < nCount; i++ )
        {
            for ( int j = 0; j < nColumnCount; j++ )
            {
                object strCell = axTree1.Items.get_CellCaption(axTree1.Items.get_SelectedItem(i),
j);
                strData +=  ( strCell != null ? strCell.ToString() : "" ) + "\t";
            }
            strData += "\r\n";
        }
        e.data.SetData( strData, EXTREELib.exClipboardFormatEnum.exCFText );
    }
}
```

The following VFP sample copies the selected items to the clipboard, as soon as the user starts dragging the items:

```
*** ActiveX Control Event ***
LPARAMETERS data, allowedeffects

local sData, nColumnCount, i, j
with thisform.Tree1.Items
    if ( .SelectCount() > 0 )
        allowedeffects = 1 && exOLEDropEffectCopy
        sData = ""
        nColumnCount = thisform.Tree1.Columns.Count
        for i = 0 to .SelectCount - 1
            for j = 0 to nColumnCount
                sData = sData + .CellCaption( .SelectedItem(i), j ) + chr(9)
            next
            sData = sData + chr(10)+ chr(13)
        next
        data.SetData( sData, 1 ) && exCFText
    endif
```

endwith

event 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 horizontal scroll bar has changed.
NewVal as Long	A long value that indicates the new scroll bar value.

If the control has no scroll bars the OffsetChanged and OversizeChanged events are not fired. When the scroll bar range is changed the OversizeChanged event is fired. Use the ScrollBars property of the control to determine which scroll bars are visible within the control. The control fires the LayoutChanged event when the user resizes a column, or change its position. Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part.

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,
AxEXTREELib._ITreeEvents_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:
AxEXTREELib._ITreeEvents_OversizeChangedEvent);
begin
end;
```

Powe…

```
begin event OversizeChanged(boolean Horizontal,long NewVal)
end event OversizeChanged
```

VB.NET

```
Private Sub OversizeChanged(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_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(oTree,Horizontal,NewVal)
RETURN
```

Syntax for OversizeChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OversizeChanged(Horizontal,NewVal)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function OversizeChanged(Horizontal,NewVal)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOversizeChanged Boolean llHorizontal Integer llNewVal
    Forward Send OnComOversizeChanged llHorizontal llNewVal
End_Procedure
```

Visual Objects

```
METHOD OCX_OversizeChanged(Horizontal,NewVal) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OversizeChanged(boolean _Horizontal,int _NewVal)
{
}
```

XBasic

```
function OversizeChanged as v (Horizontal as L,NewVal as N)
end function
```

dBASE

```
function nativeObject_OversizeChanged(Horizontal,NewVal)
return
```

event RClick ()

Fired when right mouse button is clicked.

Type	Description

Use the RClick event to add your context menu. The RClick event notifies your application when the user right clicks the control. Use the Click event to notify your application that the user clicks the control (using the left mouse button). Use the MouseDown or MouseUp event if you require the cursor position during the RClick event. Use the RClickSelect property to specify whether the user can select items by right clicking the mouse. Use the ItemFromPoint property to get the item from point. Use the ColumnFromPoint property to get the column from point.

Syntax for RClick event, **/NET** version, on:

C#

```
private void RClick(object sender)
{
}
```

VB

```
Private Sub RClick(ByVal sender As System.Object) Handles RClick
End Sub
```

Syntax for RClick event, **/COM** version, on:

C#

```
private void RClick(object sender, EventArgs e)
{
}
```

C++

```
void OnRClick()
{
}
```

C++ Builder

```
void __fastcall RClick(TObject *Sender)
{
}
```

Delphi

```
procedure RClick(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure RClick(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event RClick()
end event RClick
```

VB.NET

```
Private Sub RClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RClick
End Sub
```

VB6

```
Private Sub RClick()
End Sub
```

VBA

```
Private Sub RClick()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnRClick(oTree)
RETURN
```

Syntax for RClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="RClick()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function RClick()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComRClick
    Forward Send OnComRClick
End_Procedure
```

Visual Objects

```
METHOD OCX_RClick() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_RClick()
{
}
```

XBasic

```
function RClick as v ()
end function
```

dBASE

```
function nativeObject_RClick()
return
```

The following VB sample use Exontrol's ExPopupMenu Component to display a context menu when user has clicked the right mouse button in the control's client area:

```
Private Sub Tree1_RClick()
   Dim i As Long
   i = PopupMenu1.ShowAtCursor
End Sub
```

If you need to add a context menu based on the item you can use the MouseUp event, like in the following VB sample:

```
Private Sub Tree1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   If (Button = 2) Then
      ' Converts the container coordinates to client coordinates
      X = X / Screen.TwipsPerPixelX
      Y = Y / Screen.TwipsPerPixelY
      Dim h As HITEM
      Dim c As Long, hit as Long
      ' Gets the item from (X,Y)
      h = Tree1.ItemFromPoint(X, Y, c, hit)
      If Not (h = 0) Then
         Dim i As Long
         PopupMenu1.Items.Add Tree1.Items.CellCaption(h, c)
```

```
        i = PopupMenu1.ShowAtCursor
      End If
   End If
End Sub
```

The following VC sample displays the caption of the cell where the mouse is released:

```
#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 OnMouseUpTree1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, hItem = m_tree.GetItemFromPoint( X, Y, &c, &hit );
   if ( ( hItem != 0 ) || ( c != 0 ) )
   {
      CItems items = m_tree.GetItems();
      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 displays the caption of the cell where the mouse is released:

```
Private Sub AxTree1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXTREELib._ITreeEvents_MouseUpEvent) Handles AxTree1.MouseUpEvent
   With AxTree1
      Dim i As Integer, c As Integer, hit As EXTREELib.HitTestInfoEnum
      i = .get_ItemFromPoint(e.x, e.y, c, hit)
      If (Not (i = 0) Or Not (c = 0)) Then
         Debug.WriteLine("Cell: " & .Items.CellCaption(i, c) & " Hit: " & hit.ToString())
      End If
   End With
End Sub
```

The following C# sample displays the caption of the cell where the mouse is released:

```
private void axTree1_MouseUpEvent(object sender,
AxEXTREELib._ITreeEvents_MouseUpEvent e)
{
   int c = 0;
   EXTREELib.HitTestInfoEnum hit;
   int i = axTree1.get_ItemFromPoint( e.x, e.y, out c,out hit );
   if ( ( i != 0 ) || ( c != 0 ) )
   {
      string s = axTree1.Items.get_CellCaption( i,c ).ToString();
      s = "Cell: " + s + ", Hit: " + hit.ToString();
      System.Diagnostics.Debug.WriteLine( s );
   }
}
```

The following VFP sample displays the caption of the cell where the mouse is released:

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

local c, hit
c = 0
hit = 0
with thisform.Tree1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
```

```
      wait window nowait .Items.CellCaption( 0, c ) + " " + Str( hit )
   endif
endwith
```

event RemoveColumn (Column as Column)

Fired before deleting a column.

Type	Description
Column as Column	A Column object being removed.

The RemoveColumn event is invoked when the control is about to remove a column. Use the RemoveColumn event to release any extra data associated to the column. Use the Remove method to remove a specific column from Columns collection. Use the Clear method to clear the columns collection. Use the RemoveItem method to remove an item. Use the RemoveAllItems method to remove all items. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item. Use the Data property to assign an extra data to a column.

Syntax for RemoveColumn event, **/NET** version, on:

C#
```
private void RemoveColumn(object sender,exontrol.EXTREELib.Column Column)
{
}
```

VB
```
Private Sub RemoveColumn(ByVal sender As System.Object,ByVal Column As exontrol.EXTREELib.Column) Handles RemoveColumn
End Sub
```

Syntax for RemoveColumn event, **/COM** version, on:

C#
```
private void RemoveColumn(object sender,
AxEXTREELib._ITreeEvents_RemoveColumnEvent e)
{
}
```

C++
```
void OnRemoveColumn(LPDISPATCH Column)
{
}
```

C++ Builder
```
void __fastcall RemoveColumn(TObject *Sender,Extreelib_tlb::IColumn *Column)
{
}
```

Delphi

```
procedure RemoveColumn(ASender: TObject; Column : IColumn);
begin
end;
```

Delphi 8 (.NET only)

```
procedure RemoveColumn(sender: System.Object; e:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_RemoveColumnEvent) Handles RemoveColumn
End Sub
```

VB6

```
Private Sub RemoveColumn(ByVal Column As EXTREELibCtl.IColumn)
End Sub
```

VBA

```
Private Sub RemoveColumn(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnRemoveColumn(oTree,Column)
RETURN
```

Syntax for RemoveColumn event, ***/COM*** version (others), on:

Java…

```
<SCRIPT EVENT="RemoveColumn(Column)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function RemoveColumn(Column)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComRemoveColumn Variant llColumn
    Forward Send OnComRemoveColumn llColumn
End_Procedure
```

Visual Objects

```
METHOD OCX_RemoveColumn(Column) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_RemoveColumn(COM _Column)
{
}
```

XBasic

```
function RemoveColumn as v (Column as OLE::Exontrol.Tree.1::IColumn)
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 indicates the handle of the item being removed.

Use the RemoveItem to release any extra data that you might have used. The control fires the RemoveItem event before removing the item. Use the RemoveItem method to remove an item from Items collection. Use the RemoveAllItems method to clear the items collection. Use the Remove method to remove a column. Use the Clear method to clear the columns collection. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item. Use the Data property to assign an extra data to a column.

Syntax for RemoveItem event, **/NET** version, on:

C#

```
private void RemoveItem(object sender,int 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,
AxEXTREELib._ITreeEvents_RemoveItemEvent e)
{
}
```

C++

```
void OnRemoveItem(long Item)
{
}
```

C++ Builder

```
void __fastcall RemoveItem(TObject *Sender,Extreelib_tlb::HITEM Item)
{
}
```

Delphi
```
procedure RemoveItem(ASender: TObject; Item : HITEM);
begin
end;
```

Delphi 8 (.NET only)
```
procedure RemoveItem(sender: System.Object; e:
AxEXTREELib._ITreeEvents_RemoveItemEvent);
begin
end;
```

Powe…
```
begin event RemoveItem(long Item)
end event RemoveItem
```

VB.NET
```
Private Sub RemoveItem(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_RemoveItemEvent) Handles RemoveItem
End Sub
```

VB6
```
Private Sub RemoveItem(ByVal Item As EXTREELibCtl.HITEM)
End Sub
```

VBA
```
Private Sub RemoveItem(ByVal Item As Long)
End Sub
```

VFP
```
LPARAMETERS Item
```

Xbas…
```
PROCEDURE OnRemoveItem(oTree,Item)
RETURN
```

Syntax for RemoveItem event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="RemoveItem(Item)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function RemoveItem(Item)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComRemoveItem HITEM llItem
   Forward Send OnComRemoveItem llItem
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.Tree.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 scroll bar being clicked.
ScrollPart as ScrollPartEnum	A ScrollPartEnum expression that indicates the part of the scroll being clicked.

Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar. The ScrollButtonClick event is fired when the user clicks and releases the mouse over an enabled part of the scroll bar. Use the ScrollBars property to specify the visible scrollbars in the control. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. Use the ScrollPartEnable property to specify enable or disable parts in the control's scrollbar. Use the ScrolPartCaption property to specify the caption of the scroll's part. Use the OffsetChanged event to notify your application that the scroll position is changed. Use the OversizeChanged event to notify your application whether the range for a specified scroll bar is changed. Use the ScrollPos property to specify the position for the control's scroll bar. Use the Background property to change the visual appearance for any part in the control's scroll bar.

Syntax for ScrollButtonClick event, **/NET** version, on:

C#
```
private void ScrollButtonClick(object sender,exontrol.EXTREELib.ScrollBarEnum
ScrollBar,exontrol.EXTREELib.ScrollPartEnum ScrollPart)
{
}
```

VB
```
Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As
exontrol.EXTREELib.ScrollBarEnum,ByVal ScrollPart As
exontrol.EXTREELib.ScrollPartEnum) Handles ScrollButtonClick
End Sub
```

Syntax for ScrollButtonClick event, **/COM** version, on:

C#
```
private void ScrollButtonClick(object sender,
AxEXTREELib._ITreeEvents_ScrollButtonClickEvent e)
{
}
```

C++

```
void OnScrollButtonClick(long ScrollBar,long ScrollPart)
{
}
```

C++ Builder

```
void __fastcall ScrollButtonClick(TObject *Sender,Extreelib_tlb::ScrollBarEnum
ScrollBar,Extreelib_tlb::ScrollPartEnum ScrollPart)
{
}
```

Delphi

```
procedure ScrollButtonClick(ASender: TObject; ScrollBar :
ScrollBarEnum;ScrollPart : ScrollPartEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ScrollButtonClick(sender: System.Object; e:
AxEXTREELib._ITreeEvents_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
AxEXTREELib._ITreeEvents_ScrollButtonClickEvent) Handles ScrollButtonClick
End Sub
```

VB6

```
Private Sub ScrollButtonClick(ByVal ScrollBar As EXTREELibCtl.ScrollBarEnum,ByVal
ScrollPart As EXTREELibCtl.ScrollPartEnum)
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart

```

Xbas…

```
PROCEDURE OnScrollButtonClick(oTree,ScrollBar,ScrollPart)
RETURN
```

Syntax for ScrollButtonClick event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="ScrollButtonClick(ScrollBar,ScrollPart)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function ScrollButtonClick(ScrollBar,ScrollPart)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComScrollButtonClick OLEScrollBarEnum llScrollBar
OLEScrollPartEnum llScrollPart
    Forward Send OnComScrollButtonClick llScrollBar llScrollPart
End_Procedure
```

Visual Objects
```
METHOD OCX_ScrollButtonClick(ScrollBar,ScrollPart) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_ScrollButtonClick(int _ScrollBar,int _ScrollPart)
{
}
```

XBasic
```
function ScrollButtonClick as v (ScrollBar as
OLE::Exontrol.Tree.1::ScrollBarEnum,ScrollPart as
OLE::Exontrol.Tree.1::ScrollPartEnum)
end function
```

dBASE
```
function nativeObject_ScrollButtonClick(ScrollBar,ScrollPart)
return
```

The following VB sample displays the identifier of the scroll's button being clicked:

```
With Tree1
   .BeginUpdate
      .ScrollBars = exDisableBoth
      .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
      .ScrollPartCaption(exVScroll, exLeftB1Part) = "<img></img>1"
```

```
    .ScrollPartCaption(exVScroll, exRightB1Part) = "<img></img>2"
  .EndUpdate
End With
```

```
Private Sub Tree1_ScrollButtonClick(ByVal ScrollPart As EXTREELibCtl.ScrollPartEnum)
  MsgBox (ScrollPart)
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxTree1
  .BeginUpdate()
  .ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth
  .set_ScrollPartVisible(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part Or EXTREELib.ScrollPartEnum.exRightB1Part, True)
  .set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
  .set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exRightB1Part, "<img></img>2")
  .EndUpdate()
End With
```

```
Private Sub AxTree1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As
AxEXTREELib._ITreeEvents_ScrollButtonClickEvent) Handles AxTree1.ScrollButtonClick
    MessageBox.Show( e.scrollPart.ToString())
End Sub
```

The following C# sample displays the identifier of the scroll's button being clicked:

```
axTree1.BeginUpdate();
axTree1.ScrollBars = EXTREELib.ScrollBarsEnum.exDisableBoth;
axTree1.set_ScrollPartVisible(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part | EXTREELib.ScrollPartEnum.exRightB1Part, true);
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axTree1.set_ScrollPartCaption(EXTREELib.ScrollBarEnum.exVScroll,
EXTREELib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axTree1.EndUpdate();
```

```
private void axTree1_ScrollButtonClick(object sender,
AxEXTREELib._ITreeEvents_ScrollButtonClickEvent e)
{
   MessageBox.Show(e.scrollPart.ToString());
}
```

The following C++ sample displays the identifier of the scroll's button being clicked:

```
m_tree.BeginUpdate();
m_tree.SetScrollBars( 15 /*exDisableBoth*/ );
m_tree.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_tree.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_tree.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2")
);
m_tree.EndUpdate();
```

```
void OnScrollButtonClickTree1(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.Tree1
   .BeginUpdate
      .ScrollBars = 15
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "<img></img>1"
      .ScrollPartCaption(0, 32) = "<img></img>2"
   .EndUpdate
EndWith
```

```
*** ActiveX Control Event ***
LPARAMETERS scrollpart
```

```
wait window nowait ltrim(str(scrollpart))
```

event SelectionChanged ()

Fired after a new item has been selected.

Type	Description

Use the SelectionChanged event to notify your application that the user selects an item (that's selectable). Use the SelectableItem property to specify the user can select an item. The control supports single or multiple selection as well. When an item is selected or unselected the control fires the SelectionChanged event. Use the SingleSel property to specify if your control supports single or multiple selection. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the SelectItem to select or unselect a specified item. Use the FocusItem property to get the focused item. If the control supports only single selection, you can use the FocusItem property to get the selected/focused item because they are always the same. Use the SelForeColor and SelBackColor properties to specify colors for selected items.

Syntax for SelectionChanged event, **/NET** version, on:

C#

```
private void SelectionChanged(object sender)
{
}
```

VB

```
Private Sub SelectionChanged(ByVal sender As System.Object) Handles
SelectionChanged
End Sub
```

Syntax for SelectionChanged event, **/COM** version, on:

C#

```
private void SelectionChanged(object sender, EventArgs e)
{
}
```

C++

```
void OnSelectionChanged()
{
}
```

C++ Builder

```
void __fastcall SelectionChanged(TObject *Sender)
{
}
```

Delphi

```
procedure SelectionChanged(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure SelectionChanged(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event SelectionChanged()
end event SelectionChanged
```

VB.NET

```
Private Sub SelectionChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SelectionChanged
End Sub
```

VB6

```
Private Sub SelectionChanged()
End Sub
```

VBA

```
Private Sub SelectionChanged()
End Sub
```

VFP

```
LPARAMETERS nop

```

Xbas…

```
PROCEDURE OnSelectionChanged(oTree)
RETURN
```

Syntax for SelectionChanged event, ***/COM*** version (others), on:

Java…

```
<SCRIPT EVENT="SelectionChanged()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function SelectionChanged()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComSelectionChanged
    Forward Send OnComSelectionChanged
End_Procedure
```

Visual Objects

```
METHOD OCX_SelectionChanged() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_SelectionChanged()
{
}
```

XBasic

```
function SelectionChanged as v ()
end function
```

dBASE

```
function nativeObject_SelectionChanged()
return
```

The following VB sample displays the selected items:

```
Private Sub Tree1_SelectionChanged()
   On Error Resume Next
   Dim h As HITEM
   Dim i As Long, j As Long, nCols As Long, nSels As Long
   nCols = Tree1.Columns.Count
   With Tree1.Items
   nSels = .SelectCount
   For i = 0 To nSels - 1
      Dim s As String
      For j = 0 To nCols - 1
         s = s + .CellCaption(.SelectedItem(i), j) + Chr(9)
      Next
   Debug.Print s
   Next
   End With
End Sub
```

The following VB sample expands programmatically items when the selection is changed:

```
Private Sub Tree1_SelectionChanged()
   Tree1.Items.ExpandItem(Tree1.Items.SelectedItem()) = True
End Sub
```

The following VB sample displays the selected items:

```
Private Sub Tree1_SelectionChanged()
   Dim i As Long
   With Tree1.Items
      For i = 0 To .SelectCount - 1
         Debug.Print .CellCaption(.SelectedItem(i), 0)
      Next
   End With
End Sub
```

The following VC sample displays the selected items:

```
#include "Items.h"

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}

void OnSelectionChangedTree1()
{
   CItems items = m_tree.GetItems();
   for ( long i = 0; i < items.GetSelectCount(); i++ )
   {
```

```
        COleVariant vtItem( items.GetSelectedItem( i ) );
        CString strOutput;
        strOutput.Format( "%s\n", V2S( &items.GetCellCaption( vtItem, COleVariant( (long)0 )
) ) );
        OutputDebugString( strOutput );
    }
}
```

The following VB.NET sample displays the selected items:

```
Private Sub AxTree1_SelectionChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxTree1.SelectionChanged
    With AxTree1.Items
        Dim i As Integer
        For i = 0 To .SelectCount - 1
            Debug.WriteLine(.CellCaption(.SelectedItem(i), 0))
        Next
    End With
End Sub
```

The following C# sample displays the selected items:

```
private void axTree1_SelectionChanged(object sender, System.EventArgs e)
{
    for ( int i = 0; i < axTree1.Items.SelectCount - 1; i++ )
    {
        object cell = axTree1.Items.get_CellCaption( axTree1.Items.get_SelectedItem( i), 0 );
        System.Diagnostics.Debug.WriteLine( cell != null ? cell.ToString() : "" );
    }
}
```

The following VFP sample displays the selected items:

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

with thisform.Tree1.Items
    for i = 0 to .SelectCount - 1
        .DefaultItem = .SelectedItem( i )
        wait window nowait .CellCaption( 0, 0 )
```

```
    next
endwith
```

event Sort ()

Occurs when the control sorts a column.

Type	Description

The control fires the Sort event when the control sorts a column (the user clicks the column's head) or when the sorting position is changed in the control's sort bar. Use the SortOnClick property to specify the action that control executes when the user clicks the column's head. Use the SortBarVisible property to show the control's sort bar. Use the SortOrder property to sorts a column at runtime. Use the SortPosition property to determine the position of the column in the sorting columns collection. Use the ItemBySortPosition property to access a column giving its position in the sorting columns collection. Use the Sort event to sort the data when the SortOnClk property is exUserSort. Use the SingleSort property to allow sorting by single or multiple columns.

Syntax for Sort event, **/NET** version, on:

C#
```
private void Sort(object sender)
{
}
```

VB
```
Private Sub Sort(ByVal sender As System.Object) Handles Sort
End Sub
```

Syntax for Sort event, **/COM** version, on:

C#
```
private void Sort(object sender, EventArgs e)
{
}
```

C++
```
void OnSort()
{
}
```

C++ Builder
```
void __fastcall Sort(TObject *Sender)
{
}
```

Delphi
```
procedure Sort(ASender: TObject; );
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure Sort(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event Sort()
end event Sort
```

VB.NET

```
Private Sub Sort(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Sort
End Sub
```

VB6

```
Private Sub Sort()
End Sub
```

VBA

```
Private Sub Sort()
End Sub
```

VFP

```
LPARAMETERS nop

```

Xbas…

```
PROCEDURE OnSort(oTree)
RETURN
```

Syntax for Sort event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Sort()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Sort()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComSort
   Forward Send OnComSort
End_Procedure
```

Visual Objects

```
METHOD OCX_Sort() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Sort()
{
}
```

XBasic

```
function Sort as v ()
end function
```

dBASE

```
function nativeObject_Sort()
return
```

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

```
Private Sub Tree1_Sort()
    Dim s As String, i As Long, c As Column
    i = 0
    With Tree1.Columns
        Set c = .ItemBySortPosition(i)
        While (Not c Is Nothing)
            s = s + """" & c.Caption & """ " & IIf(c.SortOrder = SortAscending, "A", "D") & " "
            i = i + 1
            Set c = .ItemBySortPosition(i)
        Wend
    End With
    s = "Sort: " & s
    Debug.Print s
End Sub
```

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

```
void OnSortTree1()
{
    CString strOutput;
    CColumns columns = m_tree.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 AxTree1_Sort(ByVal sender As Object, ByVal e As System.EventArgs) Handles
AxTree1.Sort
   With AxTree1
      Dim s As String, i As Integer, c As EXTREELib.Column
      i = 0
      With AxTree1.Columns
         c = .ItemBySortPosition(i)
         While (Not c Is Nothing)
            s = s + """" & c.Caption & """ " & IIf(c.SortOrder =
EXTREELib.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 axTree1_Sort(object sender, System.EventArgs e)
{
   string strOutput = "";
   int i = 0;
   EXTREELib.Column column = axTree1.Columns.get_ItemBySortPosition( i );
```

```
    while ( column != null )
    {
        strOutput += column.Caption + " " + ( column.SortOrder ==
EXTREELib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
        column = axTree1.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.Tree1.Columns
    c = .ItemBySortPosition(i)
    do While (!isnull(c))
        with c
            s = s + "'" + .Caption
            s = s + "' " + Iif(.SortOrder = 1, "A", "D") + " "
            i = i + 1
        endwith
        c = .ItemBySortPosition(i)
    enddo
endwith
s = "Sort: " + s
wait window nowait s
```

event ToolTip (Item as HITEM, ColIndex as Long, ByRef Visible as Boolean, ByRef X as Long, ByRef 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	(By Reference) A boolean expression that indicates whether the object's tooltip is visible.
X as Long	(By Reference) A long expression that indicates the left location of the tooltip window. The x values is always expressed in screen coordinates.
Y as Long	(By Reference) A long expression that indicates the top location of the tooltip window. The y values is always expressed in screen coordinates.
CX as Long	A 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.

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
End Sub
```

Syntax for ToolTip event, **/COM** version, on:

C#
```
private void ToolTip(object sender, AxEXTREELib._ITreeEvents_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,Extreelib_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: AxEXTREELib._ITreeEvents_ToolTipEvent);
begin
end;
```

Powe…
```
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 AxEXTREELib._ITreeEvents_ToolTipEvent) Handles ToolTip
End Sub
```

VB6
```
Private Sub ToolTip(ByVal Item As EXTREELibCtl.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(oTree,Item,ColIndex,Visible,X,Y,CX,CY)
RETURN
```

Syntax for ToolTip event, ***/COM*** version (others), on:

Java…

```
<SCRIPT EVENT="ToolTip(Item,ColIndex,Visible,X,Y,CX,CY)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ToolTip(Item,ColIndex,Visible,X,Y,CX,CY)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComToolTip HITEM llItem Integer llColIndex Boolean llVisible Integer
llX Integer llY Integer llCX Integer llCY
    Forward Send OnComToolTip llItem llColIndex llVisible llX llY llCX llCY
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.Tree.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
```

Expressions

An expression is a string which defines a formula or criteria, that's evaluated at runtime. The expression may be a combination of variables, constants, strings, dates and operators/functions. For instance 1000 format `` gets 1,000.00 for US format, while 1.000,00 is displayed for German format.

The Exontrol's **eXPression** component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

Usage examples:

- 100 + 200, adds two numbers and returns 300
- "100" + 200, concatenates the strings, and returns "100200"
- currency(1000) displays the value in currency format based on the current regional setting, such as "$1,000.00" for US format.
- 1000 format `` gets 1,000.00 for English format, while 1.000,00 is displayed for German format
- 1000 format `2|.|3|,` always gets 1,000.00 no matter of settings in the control panel.
- date(value) format `MMM d, yyyy` , returns the date such as Sep 2, 2023, for English format
- upper("string") converts the giving string in uppercase letters, such as "STRING"
- date(dateS('3/1/' + year(9:=#1/1/2018#)) + ((1:=(((255 - 11 * (year(=:9) mod 19)) - 21) mod 30) + 21) + (=:1 > 48 ? -1 : 0) + 6 - ((year(=:9) + int(year(=:9) / 4)) + =:1 + (=:1 > 48 ? -1 : 0) + 1) mod 7)) returns the date the Easter Sunday will fall, for year 2018. In this case the expression returns #4/1/2018#. If #1/1/2018# is replaced with #1/1/2019#, the expression returns #4/21/2019#.

Listed bellow are all predefined constants, operators and functions the general-expression supports:

The constants can be represented as:

- numbers in **decimal** format (where dot character specifies the decimal separator). For instance: -1, 100, 20.45, .99 and so on
- numbers in **hexa-decimal** format (preceded by **0x** or **0X** sequence), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and

programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, 0xFF, 0x00FF00, and so so.
- **date-time** in format **#mm/dd/yyyy hh:mm:ss#**, For instance, #1/31/2001 10:00# means the January 31th, 2001, 10:00 AM
- **string**, if it starts / ends with any of the **'** or ` or **"** characters. If you require the starting character inside the string, it should be escaped ( preceded by a \ character ). For instance, `Mihai`, "Filimon", 'has', "\"a quote\"", and so on

The predefined constants are:

- **bias** (BIAS constant), defines the difference, in minutes, between Coordinated Universal Time (UTC) and local time. For example, Middle European Time (MET, GMT+01:00) has a time zone bias of "-60" because it is one hour ahead of UTC. Pacific Standard Time (PST, GMT-08:00) has a time zone bias of "+480" because it is eight hours behind UTC. For instance, date(value - bias/24/60) converts the UTC time to local time, or date(date('now') + bias/24/60) converts the current local time to UTC time. For instance, "date(value - bias/24/60)" converts the value date-time from UTC to local time, while "date(value + bias/24/60)" converts the local-time to UTC time.
- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpi returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpix returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- ***** (multiplicity operator), priority 5
- **/** (divide operator), priority 5
- **mod** (reminder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

The supported binary boolean operators, all these with the same priority 0, are :

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

The supported binary range operators, all these with the same priority 5, are :

- a **MIN** b (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression value MIN 10 returns always a value greater than 10.
- a **MAX** b (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression value MAX 100 returns always a value less than 100.

The supported binary operators, all these with the same priority 0, are :

- **:= (Store operator),** stores the result of expression to variable. The syntax for := operator is

  variable := expression

  where variable is a integer between 0 and 9. You can use the **=:** operator to restore any stored variable (please make the difference between := and =:). For instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

- **=: (Restore operator),** restores the giving variable (previously saved using the store operator). The syntax for =: operator is

  =: variable

where variable is a integer between 0 and 9. You can use the **:=** operator to store the value of any expression (please make the difference between := and =:). For instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

The supported ternary operators, all these with the same priority 0, are :

- **?** (**Immediate If operator**), returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for *?* operator is

expression ? true_part : false_part

, while it executes and returns the true_part if the expression is true, else it executes and returns the false_part. For instance, the *%0 = 1 ? 'One' : (%0 = 2 ? 'Two' : 'not found')* returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the case() statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- ***array*** *(at operator),* returns the element from an array giving its index (0 base). The *array* operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for *array* operator is

expression array (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *month(value)-1 array ('J','F','M','A','M','Jun','J','A','S','O','N','D')* is equivalent with *month(value)-1 case (default:''; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D')*.

- ***in*** *(include operator),* specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

expression in (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *value in (11,22,33,44,13)* is equivalent with *(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)*. The *in* operator is not a time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the

in operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- ***switch*** *(switch operator),* returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker that iif (immediate if operator) alterative.

- ***case()*** *(case operator)* returns and executes one of n expressions, depending on the evaluation of the expression (IIF - immediate IF operator is a binary case() operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the case() operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, ... the default_expression is executed and returned. If the value of the expression is c1, then the *case()* operator executes and returns the *expression1.* The *default, c1, c2, c3, ...* must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)* indicates that only *#1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002#* dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

  - #4/1/2009#, from hours 06:00 AM to 12:00 PM
  - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM,

04:00PM, 06:00PM and 10:00PM
- #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using iif and or expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. The type operator may return any of the following: 0 - empty (not initialized), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, **7 - date**, **8 - string**, 9 - object, 10 - error, **11 - boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bites. For instance *type(%1) = 8* specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, hex(`FF`) returns 255, while the hex(255) or hex(0xFF) returns the `FF` string. The hex(hex(`FFFFFFFF`)) always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitand 0x10111000 returns 0x00001000.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitor 0x10111000 returns 0x11111000.
- a **bitxor** b (binary operator) computes the XOR (exclusive-OR) operation on bits of a and b, and returns the unsigned value. For instance, 0x01110010 bitxor 0x10101010 returns 0x11011000.

- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, 128 bitshift 1 returns 64 (dividing by 2) or 128 bitshift (-1) returns 256 (multiplying by 2)
- **bitnot** (unary operator) flips every bit of x, and returns the unsigned value. For instance, bitnot(0x00FF0000) returns 0xFF00FFFF.

The operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as $1,000.00, for US format.
- value **format** 'flags' (binary operator) formats a numeric value with specified flags. The format method formats numeric or date expressions (depends on the type of the value, explained at operators for dates). If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the "*1000 format ""* displays 1,000.00 for English format, while 1.000,00 is displayed for German format. "1000 format '2|.|3|,'" will always displays 1,000.00 no matter of the settings in your control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

  The ' flags' for format operator is a list of values separated by | character such as '*NumDigits*|*DecimalSep*|*Grouping*|*ThousandSep*|*NegativeOrder*|*LeadingZero*' with the following meanings:

  - *NumDigits* - specifies the number of fractional digits, If the flag is missing, the

field "No. of digits after decimal" from "Regional and Language Options" is using.
    - *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
    - Grouping - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
    - *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
    - *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
        - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
        - 1 - Negative sign, number; for example, -1.1
        - 2 - Negative sign, space, number; for example, - 1.1
        - 3 - Number, negative sign; for example, 1.1-
        - 4 - Number, space, negative sign; for example, 1.1 -
    - *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

The operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"

- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahiM"
- a **startwith** b (binary operator) specifies whether a string starts with specified string (0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1
- a **endwith** b (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- a **contains** b (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- a **left** b (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- a **right** b (binary operator) retrieves the right part of the string. For instance *"Mihai" right 2* returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" lfind "C"* returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" rfind "C"* returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance *"Mihai" mid 2* returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance *"Mihai" count "i"* returns 2.
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result. For instance, the *"Mihai" replace "i" with ""* returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b (binary operator) splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.
- a **like** b (binary operator) compares the string a against the pattern b. The pattern b may contain wild-characters such as *, ?, # or [] and can have multiple patterns separated by space character. In order to have the space, or any other wild-character inside the pattern, it has to be escaped, or in other words it should be preceded by a \ character. For instance value like `F*e` matches all strings that start with F and ends on e, or value like `a* b*` indicates any strings that start with a or b character.
- a **lpad** b (binary operator) pads the value of a to the left with b padding pattern. For instance, 12 lpad "0000" generates the string "0012".
- a **rpad** b (binary operator) pads the value of a to the right with b padding pattern. For instance, 12 lpad "____" generates the string "12__".
- a **concat** b (binary operator) concatenates the a (as string) for b times. For instance, "x" concat 5, generates the string "xxxxx".

The operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the *timeF(#1/1/2001 13:00#)* returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the *shortdate(#1/1/2001 13:00#)* returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the *shortdateF(#1/1/2001 13:00#)* returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the *dateF(#01/01/2001 14:00:00#)* returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the *longdate(#1/1/2001 13:00#)* returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the *year(#12/31/1971 13:14:15#)* returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the *month(#12/31/1971 13:14:15#)* returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the *day(#12/31/1971 13:14:15#)* returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the *yearday(#12/31/1971 13:14:15#)* returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the *weekday(#12/31/1971 13:14:15#)* returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the *hour(#12/31/1971 13:14:15#)* returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the *min(#12/31/1971 13:14:15#)* returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the *sec(#12/31/1971 13:14:15#)* returns 15
- value **format** 'flags' (binary operator) formats a date expression with specified flags. The format method formats numeric (depends on the type of the value, explained at operators for numbers) or date expressions. If not supported, the value is formatted as a number (the date format is supported by newer version only). The flags specifies the format picture string that is used to form the date. Possible values for the format picture string are defined below. For instance, the *date(value) format `MMM d, yyyy`*

returns "Sep 2, 2023"

The following table defines the format types used to represent days:

- d, day of the month as digits without leading zeros for single-digit days (8)
- dd, day of the month as digits with leading zeros for single-digit days (08)
- ddd, abbreviated day of the week as specified by the current locale ("Mon" in English)
- dddd, day of the week as specified by the current locale ("Monday" in English)

The following table defines the format types used to represent months:

- M, month as digits without leading zeros for single-digit months (4)
- MM, month as digits with leading zeros for single-digit months (04)
- MMM, abbreviated month as specified by the current locale ("Nov" in English)
- MMMM, month as specified by the current locale ("November" for English)

The following table defines the format types used to represent years:

- y, year represented only by the last digit (3)
- yy, year represented only by the last two digits. A leading zero is added for single-digit years (03)
- yyy, year represented by a full four or five digits, depending on the calendar used. Thai Buddhist and Korean calendars have five-digit years. The "yyyy" pattern shows five digits for these two calendars, and four digits for all other supported calendars. Calendars that have single-digit or two-digit years, such as for the Japanese Emperor era, are represented differently. A single-digit year is represented with a leading zero, for example, "03". A two-digit year is represented with two digits, for example, "13". No additional leading zeros are displayed.
- yyyy, behaves identically to "yyyy"

The following table defines the format types used to represent era:

- g, period/era string formatted as specified by the CAL_SERASTRING value (ignored if there is no associated era or period string)
- gg, period/era string formatted as specified by the CAL_SERASTRING value (ignored if there is no associated era or period string)

The following table defines the format types used to represent hours:

- h, hours with no leading zero for single-digit hours; 12-hour clock
- hh, hours with leading zero for single-digit hours; 12-hour clock
- H, hours with no leading zero for single-digit hours; 24-hour clock

- HH, hours with leading zero for single-digit hours; 24-hour clock

The following table defines the format types used to represent minutes:

- m, minutes with no leading zero for single-digit minutes
- mm, minutes with leading zero for single-digit minutes

The following table defines the format types used to represent seconds:

- s, seconds with no leading zero for single-digit seconds
- ss, seconds with leading zero for single-digit seconds

The following table defines the format types used to represent time markers:

- t, one character time marker string, such as A or P
- tt, multi-character time marker string, such as AM or PM

The expression supports also **immediate if** (similar with iif in visual basic, or ? : in C++) ie cond ? value_true : value_false, which means that once that cond is true the value_true is used, else the value_false is used. Also, it supports variables, up to 10 from 0 to 9. For instance, 0:="Abc" means that in the variable 0 is "Abc", and =:0 means retrieves the value of the variable 0. For instance, the len(%0) ? (0:=(%1+%2) ? currency(=:0) else `` ) : `` gets the sum between second and third column in currency format if it is not zero, and only if the first column is not empty. As you can see you can use the variables to avoid computing several times the same thing (in this case the sum %1 and %2 .